Developpez

Le Mag

Edition de Octobre - Novembre 2010.
Numéro 30.
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

Éditorial

Voici un merveilleux magazine qui vous fera profiter de vos longues soirées d'hiver.
Developpez.com vous dévoile une nouvelle fois le meilleur des technologies informatiques pour votre plus grand plaisir.
Nos meilleurs articles, critiques de livres, questions/réponses, news sont à découvrir ou redécouvrir dans ce magazine. Profitez-en bien !

La rédaction

Article Mobiles

Développement Web pour mobiles – Les bases du HTML

Le langage HTML est le langage de base permettant de construire des pages Web, que celles-ci soient destinées à être affichées sur un iPhone / Android ou non.

par **Eric Sarrion**
Page 44

Article 2D/3D/Jeux

Moteur de lumières dynamiques 2D

Cet article va vous permettre de comprendre le fonctionnement du moteur de lumières dynamiques en dimension 2 conçu pour le projet Holyspirit.

par **Alexandre Laurent**
Page 56

#bijava : faut-il casser la compatibilité du langage Java ?

Le langage **Java** va bientôt fêter ses 15 ans d'existence (sa version initiale datant du *23 janvier 1996*). On peut même remonter à une vingtaine d'années si on prend en compte sa conception via le **Green Project** et le langage **Oak**.
Bien sûr le langage et l'API ont évolués entre temps, tout en respectant au mieux la sacro-sainte règle de la compatibilité ascendante.

Pour rappel il y a deux niveaux de compatibilité ascendante :

- la compatibilité *binaire*, qui veut qu'un programme compilé avec une version plus ancienne du JDK puisse fonctionner de la même manière sur une JVM plus récente ;
- la compatibilité des *sources*, qui veut qu'un programme compilable avec une version plus ancienne du JDK puisse être compilé sans erreur sur un JDK plus récent, tout en produisant une application qui fonctionne de la même manière.

Tout au long de son évolution, la plateforme **Java** a toujours fait de son mieux pour respecter ces deux niveaux de compatibilité, même si quelques sacrifices ont pu être consentis, en particulier sur la compatibilité des sources en imposant quelques légères adaptations du code dans certains cas.
Toutefois, ce n'est pas sans défaut puisque cela implique un grand nombre de limitations. Il est temps de songer à produire une version incompatible, ou tout du moins en partie...

#bijava : Repenser Java

En fait cela fait quelques temps que j'ai ce billet en tête, sans jamais avoir eu le temps d'approfondir la chose. Mais *Stephen Colebourne* a lancé les discussions sur le sujet, avec une série de billets concernant le prochain "Grand Langage" (Lien1) de la **JVM**, qu'il nomme "***#bijava***" et dont il détaille certains changements incompatibles comme la suppression des types primitifs (Lien2) et des checked-exceptions (Lien3).
En réalité c'est principalement la compatibilité des sources qui est remise en cause, car cela limite trop l'évolution du langage, ou en tout cas l'utilisation qui en est faite.
Passons au tout-objet : plus de primitives ni de tableaux
Les primitives sont traitées de manière particulière dans le langage et apportent leurs lots de restrictions et de cas particulier :

- les types primitifs ne peuvent pas être directement stockés dans un tableau d'objets, ainsi que les paramètres varargs, à moins d'utiliser uniquement un tableau de primitifs ;
- les primitifs ne sont pas utilisables avec les **Generics** ;
- les primitifs ne peuvent pas être utilisés avec la réflexion sans passer par une conversion vers les types wrapper ;
- les primitifs ne sont pas des objets et mettent en place des concepts bien distincts des références.

Il pourrait être intéressant de passer au tout-objet et de se débarrasser de cela.
Les types primitifs pourraient être remplacés par le type wrapper correspondant (quitte à ce que la JVM continue à les utiliser en interne pour des raisons de performance si besoin). À la rigueur, par simplicité, on pourrait même utiliser les types primitifs comme synonymes des types wrapper (par exemple utiliser **int** serait strictement équivalent à **Integer**, sans avoir de mécanisme d'autoboxing à l'exécution).
L'intérêt de tout cela consiste à manipuler uniquement des références sur des objets, en supprimant les cas particuliers. Cela faciliterait l'utilisation des types nullables et l'association de l'opérateur == à la méthode **equals()**.
En finir avec les checked-exceptions
Les *checked-exceptions* : un des gros concepts de **Java**, qui promettait un code plus sûr en obligeant les développeurs à traiter les exceptions. Dans le principe l'intention est louable, mais en pratique cela aboutit trop souvent à l'effet inverse : la gestion des exceptions étant tellement lourde qu'on se retrouve à traiter les exceptions uniquement pour rendre le code compilable, et pas forcément pour mettre en place une solution de contournement.
On se retrouve trop souvent avec des traitements inutiles, voire dangereux, car ils peuvent masquer l'origine exacte d'un problème en en générant d'autres par la suite. Il faut en finir avec cela, surtout qu'aucun problème d'incompatibilité ne se poserait : cette notion n'existe qu'au niveau du langage via une vérification du compilateur. Lors de l'exécution il n'y a aucune différenciation entre les deux types d'exceptions (d'ailleurs il y a moyen de gruger le compilateur pour remonter silencieusement une checked-exception (Lien4)).

Quid de la compatibilité ?

Les propositions ci-dessus pourraient très bien être effectuées en conservant la compatibilité binaire.
Mais tant qu'à repenser le langage, ne pourrait-on pas aller encore plus loin, quitte à proposer à côté un "mode de compatibilité" pour les anciennes applications. Le JDK8 devrait apporter un concept de modularisation (projet **Jigsaw**), et il fut un temps question d'un nouveau format d'archives afin de combler les défauts du format JAR.
On pourrait profiter de ces deux concepts pour mettre en place une cassure dans la compatibilité :

- les applications actuelles seraient exécutées avec des modules garantissant la compatibilité ascendante ;
- les applications futures s'exécuteraient avec des modules incompatibles, profitant des évolutions

du langage.

L'idée est simple : modifier grandement la syntaxe et l'API de **Java**, afin de repartir à zéro sur des bases un peu plus saines. Les programmes et librairies Java actuels seraient exécutés dans un "mode de compatibilité", mais les nouveaux projets pourraient bénéficier de tous les avantages de la nouvelle plateforme **Java**.

Et si on allait plus loin ?

Nota bene : il ne s'agit ici que d'idées en vrac de ma part et non pas de proposition officielle ou quoi que ce soit d'autre...

Nettoyer l'API standard

L'API standard est assez énorme, et en toute logique elle n'est pas forcément toujours parfaite. Il est temps de faire un peu le ménage là dedans et de supprimer les classes/méthodes/attributs inutiles...

Non seulement l'API conserve toujours un grand nombre d'éléments "*deprecated*" dont il serait bon de se débarrasser définitivement, mais elle possède également un grand nombre d'éléments devenus inutiles et remplacés par une nouvelle API plus propre. Je pense par exemple aux classes **Vector/Hashtable** (remplacées par l'API de collection et les implémentations de **List/Map**), à la méthode **Runtime.exec()** (remplacée par la classe **ProcessBuilder**), à la classe **File** (qui sera remplacée par la classe **Path** et son API bien plus complète dans **Java 7**).

En toute logique la suppression de ces éléments impliquerait également une fin de la compatibilité binaire.

Mais on pourrait ruser un peu en utilisant une "suppression logique", par exemple une annotation **@Deleted** qui fonctionnerait un peu sur le même principe que **@Deprecated**, à une différence près : les éléments ainsi marqués ne seraient visibles que dans le "mode de compatibilité" avec les anciens binaires.

En clair, une méthode marquée avec l'annotation **@Deleted** serait toujours visible pour les anciennes applications (de la même manière qu'avec l'annotation **@Deprecated**). Par contre elle serait totalement inexistante dans les nouvelles applications, que ce soit dans la documentation, à la compilation ou même à l'exécution.

L'intérêt étant de pouvoir faire le ménage en proposant aux futurs développeurs une API claire et nette !

Cette annotation pourrait également être associée à une version du JDK et un message d'erreur, par exemple :

```
@Deleted(version=1.7, message="Use
java.nio.file.Path instead")
public class File {
    ...
```

En compilant pour une application Java 1.7, la classe n'existe pas et son utilisation affiche le message d'erreur indiqué (afin de faciliter la portabilité de l'application). Par contre en exécutant un programme plus ancien, la classe serait parfaitement visible afin de garantir la compatibilité.

Revoir complètement les concepts associés au langage

Certains des concepts du monde **Java** sont un frein à son évolution. Je pense en particulier à la notion de getter/setter qui est utilisée massivement bien qu'il ne s'agisse que d'une convention de nommage.

Pourtant cela a soulevé de nombreuses discussions quant à l'intégration d'un mécanisme de **property** au sein du langage. En effet on y distinguait deux grandes écoles :

- des propositions de **property** simpliste basées sur du sucre syntaxique, qui se contentait en fait de générer les méthodes getter/setter, mais qui avait l'avantage de rester totalement compatible et en accord avec le concept des getter/setter ;
- des propositions plus poussées qui intégraient un vrai système de **property** dans le langage, mais qui impliquaient de revoir toutes les API afin de pouvoir en profiter pleinement.

Si on veut un vrai mécanisme de **property** en **Java**, il serait préférable de "casser" le langage.

Supprimer les tableaux (aussi)

Les tableaux posent de nombreux problèmes. Il s'agit d'objets un peu atypiques qui nécessitent des traitements particuliers à tous les niveaux. Ils possèdent un système d'héritage particulier et dangereux (les vérifications de types ne peuvent s'effectuer qu'à l'exécution, donc en générant des exceptions).

On ne devrait pas les utiliser directement. Il serait nettement plus cohérent d'utiliser une classe **Array<T>**, ce qui permettrait de manipuler les tableaux au travers d'un véritable objet, qui pourrait donc bénéficier des interfaces de l'API de Collections !

Intégrer les nouveautés du langage

Dans le même ordre d'esprit, il serait intéressant d'utiliser les nouveautés du langage dans les API existantes et non pas seulement dans les nouvelles API.

On trouve ainsi un grand nombre de méthodes utilisant un **int** en paramètre et dont les valeurs possibles sont définies dans des constantes. Tout ceci pourrait avantageusement être remplacé par des **enum**s, voir par le couple **interface/enum**.

Il en est de même pour les futures évolutions : les expressions lambda sont convertibles en type SAM, ce qui les rend inutilisables avec certains listeners d'AWT/Swing qui définissent plusieurs méthodes...

Plus généralement, les évolutions du langage apportent leurs lots de nouveaux concepts et de nouveaux patterns, qui ne sont pas forcément adaptables aux anciennes API sans tout casser. Du coup l'API standard peut sembler assez disparate d'un package à l'autre...

Revoir la gestion de la synchronisation

La gestion de base de la synchronisation est gérée dans la classe **Object** avec les méthodes de synchronisation **wait()/notify()**, puisque n'importe quel objet peut servir de lock pour la synchronisation. Personnellement je n'ai jamais compris ce principe !

Sur les neuf méthodes **public** de la classe **Object**, cinq concernent les méthodes de synchronisation, que l'on retrouvera donc dans tous les types via l'héritage (puisque tout hérite de **Object**). D'une part c'est assez troublant pour le débutant, qui est rapidement confronté à ces méthodes sans forcément les comprendre, et d'autre part cela augmente le risque d'erreur en manipulant tout et n'importe quoi comme lock.

Il serait nettement préférable de bien séparer tout cela dans une classe spécifique et ainsi clarifier l'API et l'utilisation de la synchronisation. Surtout que **Java 5.0** a introduit une API de **Lock** bien plus complète. Je ne vois strictement aucun intérêt au fait de pouvoir utiliser n'importe quelle référence comme moniteur de lock !

En clair, il serait préférable d'utiliser uniquement la classe **Lock** pour gérer la synchronisation et non plus n'importe

quel objet. Cela permettrait également d'épurer l'API de la classe **Object** (et donc dans le même temps de toutes les classes).

Distinguer les constantes des attributs

Java met en place un mécanisme de constante un peu particulier : une constante est un attribut déclaré **final**, assigné en ligne à la déclaration, et dont la valeur peut être déterminée à la compilation. Par exemple :

```
public static final String HELLO = "Hello
World!"; // Constante
public static final String HELLO =
System.getProperty("java.vm.name"); // Attribut
```

Le problème vient du fait que cela peut être fait de manière involontaire. Or, lorsqu'on utilise une constante, le compilateur remplace tous les appels explicites par la valeur de la constante. Du coup le changement de la valeur de la constante n'est reporté aux classes externes qu'après une recompilation, ce qui n'est pas toujours souhaitable.

Il serait souhaitable de forcer les développeurs à définir explicitement leurs constantes, afin de mieux comprendre les mécanismes que cela implique :

```
public const String HELLO = "Hello World!";
```

Remettre tout à plat

Il y a sûrement un grand nombre d'autres modifications incompatibles qui pourrait être apportées au langage et à l'API. D'ailleurs je ne suis pas sûr que les idées de ce billet fassent l'unanimité. L'idée principale à retenir est qu'il serait temps d'engager les discussions sur un éventuel nouveau langage **Java**, qui prendrait le temps de voir les concepts mis en place par les autres langages, afin de prendre les meilleures idées ici et là...

Bref, commençons à imaginer un **nouveau Java**, quitte à donner de grands coups de balai !

Retrouvez ce billet sur le blog de Frédéric Martini : Lien5

Les derniers tutoriels et articles

À la découverte du framework Google Collections

Les Collections Java ont un peu plus d'une douzaine d'années d'existence et s'imposent comme une des plus importantes API du monde Java. De nombreux frameworks en utilisent les fonctionnalités et les étendent. C'est notamment le cas de Google-Collections qui ajoute des évolutions intéressantes comme les prédicats, les objets Multi ou Bi, les immutables, etc. Ce document est un point de départ à la découverte des éléments clés de Google-Collections.

1. Introduction

Zip avec les sources utilisées pour cet article : tuto-google-col.zip (Lien6).

Lorsque Josh Bloch crée l'API Collections en 1997 pour l'intégrer à Java, il n'imagine sans doute pas l'importance qu'elle va prendre. Les Collections rencontrent un succès immédiat et sont largement adoptées par la communauté Java. Il faut dire que tout (ou presque) est présent dès le lancement et que, mises à part quelques évolutions importantes comme les Iterators ou les génériques, l'API n'a quasiment pas changé. C'est dire si Java-Collections a bien été pensée.

Le modèle de conception de Java-Collections est une référence. Il repose sur trois axes majeurs dont tout projet gagne à s'inspirer :

- des interfaces qui définissent les collections. L'expérience montre qu'avec les collections, encore plus qu'avec les autres classes Java, les développeurs prennent vite la bonne habitude de déclarer et d'utiliser des interfaces ;
- des implémentations qui fournissent des classes abstraites et concrètes. Ces implémentations respectent les contrats définis par les interfaces, chacune à sa manière, avec ses spécificités, justifiant qu'on utilise l'une ou l'autre ;
- des algorithmes puissants et variés qui permettent de manipuler les collections et leurs données.

Exemple de tri d'une liste

```
@Test
public void testTri() {
        List<String> amis = new Vector<String>();
        amis.add("Manon");
        amis.add("Claire");
        amis.add("Julien");
        amis.add("Thierry");
        amis.add("Martin");
        amis.add("Elodie");
        System.out.println(amis);
        // --> [Manon, Claire, Julien, Thierry,
Martin, Elodie]

        Collections.sort(amis);
        System.out.println(amis);
        // --> [Claire, Elodie, Julien, Manon,
Martin, Thierry]
}
```

Un point très important à propos de l'API Java-Collections est qu'elle est extensible. De nombreux développeurs à travers le monde en ont donc étendu les fonctionnalités. Mais ce sont, sans doute, Kevin Bourrillion et Jared Levy qui sont allés le plus loin en créant Google-Collections (Lien7).

Le framework Google-Collections est désormais inclus dans le projet Guava (Lien8). Toutefois, la partie Google-Collections, rapportée à l'ensemble des dépendances ramenées par Maven lorsqu'on ne demande QUE les collections, est la plus "importante" du projet. C'est

d'ailleurs à elle qu'est consacré cet article.

Le framework Google-Collections s'intéresse à des problématiques de bas-niveau et apporte un support fiable, permettant aux développeurs de se concentrer sur les éléments importants de leurs programmes. Tout comme Java-Collections, le framework Google-Collections propose des interfaces relativement simples, avec un nombre restreint de méthodes par interface.

2. Installation

Pour profiter des fonctionnalités du framework Google-Collections dans un programme, le plus simple est encore d'utiliser Maven. Il suffit d'ajouter une dépendance dans le fichier "pom.xml" de l'application.

Dépendance à Google-Collections dans le pom

```
<dependency>
  <groupId>com.google.collections</groupId>
  <artifactId>google-collections</artifactId>
  <version>1.0</version>
</dependency>
```

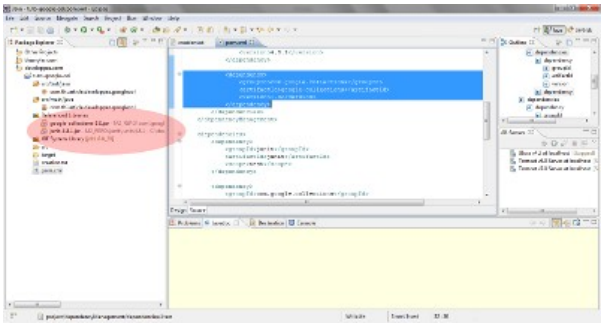
Les habitués de Maven géreront le numéro de version plus élégamment, bien que cette façon suffise largement pour cet article. Le code complet du fichier "pom.xml" utilisé pour cet article est fourni en annexe.

Puis on lance Maven depuis un terminal avec la commande suivante.

Commande Maven d'install

```
mvn clean install site eclipse:eclipse
```

Il n'y a plus qu'à importer le projet dans Eclipse. Si le projet est déjà dans le workspace Eclipse, il suffit alors de faire un refresh.



3. Déclarations rapides

Le framework Google-Collections est tout aussi vaste/riche que l'est son grand frère Java-Collections. Il est donc possible de l'aborder de différentes manières. Le parti pris de cet article est de se calquer sur l'ordre d'écriture des éléments dans un programme standard, à commencer par les déclarations.

Une des premières choses qui saute aux yeux lorsqu'on découvre Google-Collections, c'est la simplicité avec laquelle il est possible de créer des listes (*List, Set, Map*, etc.) génériques complexes sans subir les inconvénients de la syntaxe verbeuse de Java.

Voici un exemple simpliste de création de liste (*Integer*) où la syntaxe oblige à dupliquer des informations finalement inutiles.

Déclaration verbeuse d'une liste d'Integer

```
List<Integer> maListeClassique = new
ArrayList<Integer>();
...
maListeClassique.add(123);
```

Dans certains cas, moins rares qu'on ne l'imagine, les déclarations Java peuvent s'allonger dans des proportions inquiétantes. L'exemple suivant est adapté d'un cas réel. Pour le développeur, c'est systématiquement une source de doutes et de tensions.

Déclaration très verbeuse d'une map

```
Map<List<String>, Class< ? extends
List<Integer>>> maGrosseMap
                = new HashMap<List<String>,
Class< ? extends List<Integer>>>();
```

Avec Google Collections, les déclarations deviennent simples. Le typage ne se fait plus qu'à gauche du signe égal.

Déclaration simplifiée à l'aide de Google-Collections

```
import static
com.google.common.collect.Lists.newArrayList;
...
List<Integer> maListeGoogle = newArrayList();
```

À l'usage, cette écriture est vraiment pratique, plus succincte, plus claire, et il est difficile de s'en passer après y avoir goûté. Les développeurs de Google-Collections se sont appliqués à généraliser ces méthodes de création rapide pour la plupart des collections et il faut reconnaître que c'est un travail énorme qui, à lui seul, justifie l'adoption du framework.

Pour écrire une méthode static de création avec générique automatique, on peut s'inspirer du code suivant. Sun ne l'a pas généralisé pour des raisons d'homogénéité (static bad...) mais c'est bien plus élégant/pratique d'utiliser cette technique (pas seulement pour les collections) quand on en a la liberté.

Création de générique perso

```
public static <T> List<T> creerUnVectorPerso() {
        return new Vector<T>();
}

...
List<Integer> maListeMaison =
creerUnVectorPerso();
```

On notera que l'idée des déclarations simplifiées avait déjà été proposée par Josh Bloch dans son livre "Effective Java 2" et devrait arriver dans Java 7 (ou 8)

4. Filtres, prédicats et associés

Le framework Google-Collections fournit un ensemble de composants très pratiques pour filtrer, trier, comparer ou convertir des collections. Les adeptes des classes anonymes vont apprécier la suite.

4.1. Filtres

Il est fréquent de devoir filtrer des données d'une liste, issues par exemple d'une requête générique en base ou sur un web service. À l'ancienne, une telle méthode de filtre peut s'écrire comme suit.

Filtre à l'ancienne

```
static public List<Personne>
filtrerHomme1(List<Personne> personnes) {
  List<Personne> result = new
ArrayList<Personne>();
  for (Personne personne : personnes) {
    if (personne.isHomme()) {
      result.add(personne);
    }
  }

  return result
}
```

Le test JUnit suivant est proposé pour tester cette première méthode de filtre. Le début du test vise à constituer un jeu de données.

Création du jeu de données à tester

```
@Before
public void doBefore() {
        personnes = newArrayList();

        personnes.add(new Personne("Anne",
"Dupont", 27, Sexe.FEMME));
        personnes.add(new Personne("Julien",
"Lagarde", 22, Sexe.HOMME));
        personnes.add(new Personne("Manon",
"Ler", 1, Sexe.FEMME));
        personnes.add(new Personne("Mickael",
"Jordan", 48, Sexe.HOMME));
        personnes.add(new Personne("Paul",
"Berger", 65, Sexe.HOMME));
        // Pascal Dupont est le mari d'Anne
Dupont
        personnes.add(new Personne("Pascal",
"Dupont", 28, Sexe.HOMME));
        personnes.add(new Personne("Silvie",
"Alana", 15, Sexe.FEMME));
        personnes.add(new Personne("Thierry",
"Ler", 33, Sexe.HOMME));
        personnes.add(new Personne("Zoe", "Mani",
7, Sexe.FEMME));
}
```

Et la suite du test sert à valider que la méthode fonctionne bien.

Test du filtre

```
@Test
public void testTailleListe() {
        assertEquals(personnes.size(), 9);
}

@Test
public void testFiltrerHomme1() {

        List<Personne> hommes =
PersonneUtil.filtrerHomme1(personnes);
        System.out.println(hommes);
        // --> [Julien, Mickael, Pascal, Paul,
Thierry]

        assertEquals(hommes.size(), 5);

        for (Personne homme : hommes) {
                assertTrue(homme.isHomme());
        }
}
```

Les annotations @Before et @Test sont spécifiques aux tests. Une méthode annotée @Before sera lancée avant chaque test. Une méthode annotée @Test correspond à un test unitaire.

Le code de cette première méthode de filtre n'est pas très élégant. Cette méthode est composée de code technique inlassablement répété. Pourtant, bien que ce code devienne presque un pattern à force d'utilisation, on en trouve de nombreuses variantes lors des tests de qualité. En remplacement, la portion de code suivante est intéressante.

Filtre à l'aide de Iterables et Predicate

```
static public List<Personne>
filtrerHomme2(List<Personne> personnes) {
        List<Personne> result =
newArrayList(Iterables.filter(personnes,
                        new Predicate<Personne>()
{
                                public boolean
apply(Personne personne) {
                                        return
personne.isHomme();
                                }
                        }));
        return result;
}
```

Les éléments clés sont ici *Iterables.filter* et *Predicate* à propos desquels on dira quelques mots plus bas.

Évidemment, l'exemple utilisé est si simple qu'il est difficile de se rendre compte de la puissance des filtres, mis à part en imaginant un cas réel dans un projet d'entreprise. Reste bien entendu à factoriser les filtres et les *Predicates*. Et avec Java 7, on devrait avoir les *closures* et là...

Un petit détail qui a son importance, il est possible d'utiliser la classe *Collections2* à la place de *Iterables*.

Filtre à l'aide de Collections2 et Predicate

```
static public List<Personne>
filtrerHomme3(List<Personne> personnes) {
  List<Personne> result =
newArrayList(Collections2.filter(personnes,
              new Predicate<Personne>() {
    public boolean apply(Personne personne) {
      return personne.isHomme();
    }
  }));
  return result;
}
```

En effet les deux classes fournissent la méthode *filter()* et semblent fournir le même service. Mais alors quelle est la différence ? Le Javadoc de Collections2 (Lien9) et le Javadoc de Iterables (Lien10) nous en disent un peu plus sur les méthodes *filter(..)*

La doc de Collection2 nous dit : *Returns the elements of unfiltered that satisfy a predicate. The returned collection is a live view of unfiltered; changes to one affect the other. The resulting collection's iterator does not support remove(), but all other collection methods are supported. The collection's add() and addAll() methods throw an IllegalArgumentException if an element that doesn't satisfy the predicate is provided. When methods such as removeAll() and clear() are called on the filtered collection, only elements that satisfy the filter will be removed from the underlying collection. The returned collection isn't threadsafe or serializable, even if unfiltered is. Many of the filtered collection's methods, such as size(), iterate across every element in the underlying collection and determine which elements satisfy the filter. When a live view is not needed, it may be faster to copy Iterables.filter(unfiltered, predicate) and use the copy.*

La doc de Iterable nous dit : *Returns all instances of class type in unfiltered. The returned iterable has elements whose class is type or a subclass of type. The returned iterable's iterator does not support remove(). Returns an unmodifiable iterable containing all elements of the original iterable that were of the requested type*

La classe *Collections2* renvoie donc une vue *"live"*, non *thread safe*, des éléments filtrés tandis que *Iterable* renvoie une "copy". La doc le dit elle-même, si une vue *live* n'est pas nécessaire, alors l'utilisation de *Iterable* permettra d'avoir un programme plus rapide (dans la plupart des cas).

4.2. Prédicats pour les listes

Le chapitre précédent montre comment réaliser un filtre à l'aide d'un prédicat simple (homme/femme) mais les prédicats (Lien11) sont bien plus puissants. Le code suivant donne un exemple d'utilisation des méthodes de composition.

Mélange de prédicats

```
import static
com.google.common.base.Predicates.and;
import static
com.google.common.base.Predicates.or;
import static
com.google.common.base.Predicates.in;
import static
com.google.common.base.Predicates.not;
...
```

```
List<Integer> liste1 = newArrayList(1, 2, 3);
List<Integer> liste2 = newArrayList(1, 4, 5);
List<Integer> liste3 = newArrayList(1, 4, 5, 6);

@Test
public void testMelange() {
  boolean isFormuleOk1 = and(in(liste1),
in(liste2)).apply(1);
  System.out.println(isFormuleOk1); // --> true
  boolean isFormuleOk2 = and(in(liste2),
in(liste3), not(in(liste1))).apply(4);
  System.out.println(isFormuleOk2); // --> true
}
```

Ce code est si simple, comparé à toute la programmation nécessaire pour arriver au même résultat sans Google-Collections. Et encore l'exemple est volontairement simplifié et loin de représenter ce qui existe dans une vraie application d'entreprise.

L'impact est encore plus flagrant lorsqu'on s'intéresse aux mécanismes de compositions auxquels de nombreux développeurs ont été confrontés...

Composition

```
import static
com.google.common.base.Predicates.compose;
...

@Test
public void testComposition() {
  boolean isAddition = compose(in(liste3),
  new Function<Integer, Integer>() {
    public Integer apply(Integer nombre) {
      return nombre + 1;
    }
  }).apply(5);
  System.out.println(isAddition); // --> true
}
```

Une petite explication s'impose. L'utilisation de *".apply(5)"* envoie la valeur "5" à la fonction, qui l'additionne à "1" pour renvoyer la valeur "6", qui est bien dans "liste3" comme le réclame l'instruction *"in(liste3)"*. Quant à la méthode *compose()*, elle renvoie la composition d'un prédicat (ici "in") et d'une fonction. L'ensemble est un peu délicat à prendre en main mais beaucoup plus agréable à utiliser que s'il fallait s'en passer.

Retrouvez la suite de l'article de Thierry Leriche-Dessirier en ligne : Lien12

Les livres Java

Pro Android 2

Pro Android 2 shows you how to build real-world and fun mobile applications using Google's latest Android SDK. This new edition is fully updated for Android 2, covering everything from the fundamentals of building applications for embedded devices to advanced concepts such as custom 3D components, OpenGL, and touchscreens including gestures. While other Android development guides simply discuss topics, Pro Android 2 offers the combination of expert insight and real sample applications that work.

- Discover the design and architecture of the Android SDK through practical examples, and how to build mobile applications using the Android SDK.
- Explore and use the Android APIs, including those for media and Wi-Fi.
- Learn about Android 2's integrated local and web search, handwriting gesture UI, Google Translate, and text-to-speech features.

Pro Android 2 dives deep, providing you with all the knowledge and techniques you need to build mobile applications ranging from games to Google apps, including add-ons to Google Docs. You'll be able to extend and run the new Google Chrome APIs on the G1, the G2, and other next-generation Google phones and Android-enabled devices.

What you'll learn :

- How to use Android to build Java-based mobile applications for Google phones with a touch screen or keyboard (thanks to Cupcake's inclusion as of Android 1.5)
- How to design and architect using Google's latest Android SDK
- How to use the Android SDK to write mobile applications for embedded devices
- How to create 3D graphics with OpenGL and custom components
- How to build multimedia and game apps using Android's Media APIs and OpenGL
- How to use Android's location-based services, networking (Wi-Fi APIs), and security
- How to create and allow for more integrated local and web searches
- How to build handwriting gesture UIs
- How to incorporate Google Translate into your Android applications

Who is this book for ?
This book is for professional software engineers/programmers looking to move their ideas and applications into the mobile space with Android. It assumes that readers have a passable understanding of Java, including being able to write classes and handle basic inheritance structures. This book also targets hobbyists.
Par son ouverture et ses possibilités de déploiement, la plateforme Google Android basée sur Linux offre un socle et un environnement de développement puissants pour créer des applications mobiles robustes et ergonomiques. Elle met à la portée des professionnels et des particuliers la réalisation d'applications à la fois riches en fonctionnalités et adaptées aux contraintes de l'utilisation mobile.

Critique du livre par Mickaël Le Trocquer

Ce livre sur Android s'adresse surtout aux personnes ayant déjà quelques bases sur Android, ou au moins connaissant la programmation Java.
Les premiers chapitres présentent principalement le système, son découpage, sa philosophie à travers des exemples ainsi que des comparaisons à d'autres technologies (Rest, Web, POO, etc). Les chapitres suivants vont traiter plus en profondeur chacun des points abordés, et offrir, à travers de nombreux exemples, des cas d'utilisations et des explications détaillées. Tous les chapitres se terminent par un bilan succinct des notions à retenir et permet de vérifier si ces notions ont bien été comprises.
L'ensemble des points abordés est varié et permet de couvrir la plateforme Android. La version du SDK utilisée dans les explications est la 2.0 (sortie fin 2009), cette version, qui contient quelques changements importants comparée à la version précédente (1.6), introduit des concepts toujours valables dans les versions suivantes (2.1 et 2.2).
Seuls points négatifs : la séparation explication / bloc de code n'est pas suffisamment distinguable à mon goût. Le code est plus lisible lorsqu'il est encadré ou sur un fond de grisé.

Ce livre est en anglais, ce qui pourrait en rebuter plus d'un, mais il reste dans un Anglais technique très abordable, même pour quelqu'un ne maîtrisant pas complètement la langue de Shakespeare.

Pour conclure, je ne peux que recommander ce livre aux personnes souhaitant maîtriser parfaitement le développement sur Android. Ce livre permet à un développeur de partir sereinement sur un nouveau projet Android sans se poser trop de questions et ainsi de maîtriser les différentes phases d'un projet Android, de la conception à la publication sur le market.

Les EJB 3 (avec Struts 2, JSF 2, JasperReports 3, Flex 3)

Ce livre sur les EJB 3 s'adresse aux développeurs Java d'applications Web travaillant sur les frameworks Struts 2, JSF 2 ou Flex 3. Le débutant comme l'expert trouveront les informations qui leur conviennent sur l'utilisation des EJB (Enterprise JavaBeans) de manière générale et les gains de productivité apportés par la version 3.

L'auteur propose le développement avec les EJB de trois applications Web de vente en ligne aux fonctionnalités quasi identiques et qui sont basées sur des couches métier et persistance communes. À l'aide de l'IDE Eclipse et du serveur d'application JBoss 6, il exploite les fonctionnalités d'un container EJB pour :

- mettre en place une couche de persistance basée sur les Entity beans, le langage JPQL et la Java Persistence API ;
- créer des objets métiers à l'aide des Session beans et des Message-driven beans ;
- définir une politique de sécurité avec une gestion des rôles et des permissions définie dans un fichier de propriétés, une base ou un annuaire LDAP ;
- exposer des EJB 3 en tant que WebServices ;
- mettre en place des traitements différés et ponctuels à l'aide des EJB Timers ;
- faire de la programmation par aspect grâce aux Interceptors.

Tout au long des chapitres, l'auteur :

- décrit et met en place les nouveautés incluses dans les dernières versions des frameworks Struts 2 et JSF 2 ;
- détaille l'utilisation du framework GraniteDS pour réaliser la communication entre les objets Java et Flex 3 et créer une interface RIA ;
- met en avant le framework open-source de reporting JasperReports et montre son utilisation avec les EJB, Struts 2 et JSF 2 pour créer des rapports graphiques.

Enfin, l'auteur décrit les apports de la toute dernière version des EJB, la version 3.1, qui a été finalisée en décembre 2009.

Les sources des applications sont en téléchargement sur le site www.editions-eni.fr (Lien13) et l'auteur continuera de les faire évoluer sur son site.

Critique du livre par Cuisinier Gildas

Si vous souhaitez découvrir rapidement les EJB 3, ce livre est un excellent moyen de commencer.
À l'aide de captures d'écran et d'extraits de code source, il présente clairement les différents aspects de ceux-ci : la couche métier avec les Session Bean, la persistance avec JPA, la sécurité, les fonctionnalités transverses avec les interceptors ou encore l'interopérabilité par le biais des WebServices.

Mais le point fort de ce livre est qu'il ne porte pas uniquement sur les EJB, mais aussi sur leur utilisation dans des projets avec des outils et frameworks qui ne sont pas forcément liés à ceux-ci au départ.

J'ai fort apprécié la section d'intégration de Flex et des EJB, par exemple, qui est très bien expliquée, ou encore l'intégration avec JasperReport. Ces sujets sont très spécifiques et donc très rarement traités, ce qui en fait une plus-value importante (du moins pour ceux qui s'y intéresse).

Le chapitre concernant la spécification JEE 6 est appréciable également afin de survoler rapidement les nouveautés de celle-ci.

Au niveau de la forme, le livre est bien écrit et assez aisé à suivre. Mon seul regret est de devoir de temps en temps jongler entre les pages, du fait des extraits de code parfois trop verbeux.

Public
Grâce à ses codes source très bien expliqués, ses fonds d'écrans explicites et des explications très facile à comprendre, ce livre est particulièrement bien adapté à des profils néophytes désirant découvrir les EJB 3.

Retrouvez ces critiques de livres sur la page livres Java : Lien14

Performance Zend_Cache - Comparatif des performances des backends « lents »

Cet article est un comparatif des performances des deux backends lents du framework Zend. Il donne des indications pour savoir quand utiliser tel ou tel backend.

1. Généralités

Quand le code à été optimisé, un système de cache peut être nécessaire pour accélérer une application Web. Un système de cache peut ne pas être pertinent, car il oblige à utiliser des données qui ne sont potentiellement plus à jour et consomme également des ressources supplémentaires. Il est donc essentiel de mesurer l'impact et la réelle utilité d'un système de cache.

Plusieurs systèmes de cache existent : le cache au niveau du navigateur Web, les caches type proxy, les caches d'opcodes. Cet article se concentre sur le cache au niveau de l'application Web elle-même et en particulier le cache du framework PHP de Zend (Lien15).

Les backends de Zend_Cache sont divisés en deux parties :

- les caches lents : ils stockent les données dans des fichiers. Ils sont peu rapides, mais peuvent stocker des grandes quantités de données ;
- les caches rapides : ils stockent les données dans la mémoire vive. Ils sont rapides, mais sont limités par la capacité de la RAM.

Cet article se concentre sur les caches lents disponibles sans serveur spécifique (memcached ou Zend Server par exemple). Il ne s'agit donc absolument pas d'un comparatif exhaustif, mais d'un comparatif des deux grands backends : File et SQLite.

2. Méthodologie

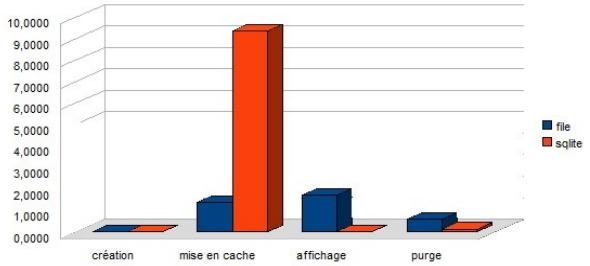
Un script PHP charge le framework Zend en **autoload**. Les temps suivants sont mesurés pour le backend File et le backend Sqlite :

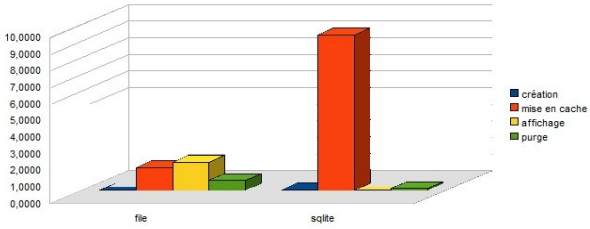
- initialisation du cache ;
- création de 100 entrées distinctes dans le cache ;
- affichage des 100 entrées créées ;
- suppression des 100 entrées de cache.

La machine de test est équipée de Windows Vista et de XAMPP. Les mesures ont été loguées dans un fichier CSV et répétées 85 fois à l'aide de l'utilitaire Apache ab, puis les moyennes on été calculées en rejetant les 5 % de valeurs aux extrêmes.

3. Résultats

Les valeurs chiffrées n'ont que peu d'intérêt. Ce qu'il est intéressant de quantifier est le rapport entre SQLite et File. La durée d'exécution en seconde est reportée sur l'axe vertical.





Comme on pouvait s'y attendre :

- l'initialisation du cache SQLite est plus lente (x3,5) ;
- la création d'une entrée de cache est beaucoup plus lente avec SQLite (x6,8) ;
- la lecture d'une entrée de cache est extrêmement plus rapide avec SQLite (x54) ;
- la suppression du cache est plus rapide avec SQLite (x4,2).

Un rapide calcul indique que si on lit plus de 4,5 fois le cache que l'on l'écrit, SQLite est plus intéressant en termes de performance. Bien sûr cette valeur peut évoluer selon le volume de données, mais l'ordre de grandeur est fixé.

Avec une durée de rafraîchissement du cache, il y a fort à parier que l'on atteigne facilement cette valeur limite de 4,5. SQLite apparaît donc comme la solution à mettre en œuvre pour un cache rapide.

4. Différences de fonctionnement File / SQLite

Le cache « File » du framework Zend écrit deux fichiers par donnée mise en cache. Le premier fichier contient les métadonnées du cache et le second les données.

Le cache « SQLite » met toutes les données mises en cache dans un seul fichier, au format spécifique optimisé SQLite. Cependant, la base peut-être verrouillée. En effet, si une donnée est en cours d'écriture, un LOCK est apposé sur le fichier SQLite et tant que l'écriture n'est pas terminée, les données ne seront pas lues et donc les données en cache ne seront pas disponibles.

Imaginons que votre application soit très volumineuse, il est possible que des écritures soient fréquentes, même si les lectures sont bien plus nombreuses. Du coup, la base étant souvent verrouillée, même si la lecture est très rapide, l'accès au cache peu globalement être plus lent.

Il est donc difficile de savoir à l'avance lequel du cache File ou du cache SQLite va être le plus rapide. La solution est de suivre la durée d'exécution de vos scripts et de tester les deux backends.

5. Supervision du temps d'exécution

La solution présentée ici utilise une sonde MRTG (Lien16) et explique l'esprit général de la solution, mais ne la détaille pas point par point.

5.1. Enregistrer les durées du script

La classe ci-dessous le permet de manière simple. Il suffit d'inclure la classe et de créer l'objet au début du script. Les méthodes magiques PHP 5 lancent le chronomètre au démarrage et sauvent les données à la fin du script.

Les informations sont stockées simplement sur une ligne d'une table MySQL.

```
<?php
/**
 *  Exemple d'utilisation developpez.com
 */
class loggeur_php
{
    private $temps;
    /**
     * Début de l'analyse     *
     */
    public function __construct()
    {
        $this->temps = microtime(true);
    }

    /**
     * Clôture de l'analyse en loggant les
informations dans la base de données
     */
    public function __destruct()
    {
        $duree = microtime(true) - $this->temps;
        $linkP = mysqli_connect('localhost',
'supervision', 'pass', 'supervision');
        $sql = "UPDATE duree SET somme=somme+
$duree, nombre=nombre+1 WHERE id=1 LIMIT 1";
        mysqli_query($linkP, $sql);
        mysqli_close($linkP);
    }
}
?>
```

5.2. La sonde MRTG

Voici un exemple de sonde MRTG, fichier à rendre exécutable.

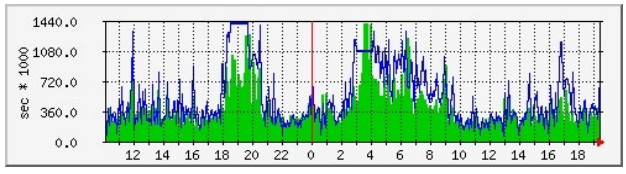
```
#!/usr/bin/php
<?php
/*
 * Permet de donner les perfs à MRTG puis les
réinitialise
 */

$linkP = mysql_connect('localhost',
'supervision', 'pass');
if ($linkP) mysql_select_db('supervision');
$sql = 'select * from performance_moyenne_v41
where id=1 limit 1';
$result = mysql_query($sql, $linkP);
while($stat = mysql_fetch_array($result))
{
    $moyenne = $stat['somme'] / $stat['nombre'];
    echo round($moyenne, 4)*1000 . "\n " .
"\n0\n0";
}

$sql2 = 'UPDATE performance_moyenne_v41 SET
somme=0, nombre=0 WHERE id=1 LIMIT 1';
mysql_query($sql2, $linkP);

mysql_close($linkP);
?>
```

5.3. Le fichier de configuration MRTG, à compléter

```
Target[***]: `/home/mrtg-sys/perf-log-mrtg.php`
PageTop[***]: <h1>Perfs</h1>
Options[***]: absolute, growright, gauge
MaxBytes[***]: 1500
Title[***]: Perfs
Ylegend[***]: sec * 1000
Legend1[***]: vitesse moyenne
Legend2[***]: ecart type
LegendI[***]: vitesse moyenne
LegendO[***]: ecart type
ShortLegend[***]: sec * 1000
```

Une fois le fichier de configuration MRTG complété, il faut régénérer le fichier index MRTG.

5.4. Les résultats



On obtient des graphiques permettant d'évaluer l'application en production avec utilisation des deux systèmes de cache.

6. Conclusion

Selon le cas, l'un ou l'autre des deux backends de cache du Zend Framework peut-être plus performant, mais il est quasiment impossible de le prévoir. Aussi, seul le suivi de son application peut permettre ce choix. Bien sûr, cela peut même servir à évaluer la vitesse de l'application sans système de cache, ou avec des serveurs de cache. Il s'agit d'une bonne pratique à intégrer dans vos projets importants.

Retrouvez l'article de Maxime Varinard en ligne : Lien17

Zend Framework 1.8 Web Application Development

The Zend Framework has a flexible architecture that lets you build modern web applications and web services easily. The MVC components make the maintenance and testing of your applications easier. However, it is not only an MVC framework for developers. It also provides an easy-to-use high-quality component library that is designed to be used the way you want, picking up specific components without requiring the use of whole framework.

It's easy to get started and produce a powerful and professional looking web site when you've got this book to hand. Taking you through a real-life application, it covers the major Zend Framework components, as well as throwing light on the best practices and design issues faced when building complex MVC applications.

This book takes you through detailed examples as well as covering the foundations you will need to get the most out of the Zend Framework. From humble beginnings you will progress through the book and slowly build upon what you have learned previously. By the end, you should have a good understanding of the Zend Framework, its components, and the issues involved in implementing a Zend Framework based application.

Create a real-life storefront application from design to deployment and explore all the major aspects of the Zend Framework

What you will learn from this book :

- Explore the features of Zend Framework's MVC architecture
- Learn about designing and implementing an MVC application
- Avoid common mistakes made when first learning an MVC framework
- Test your applications with the popular PHPUnit framework
- Interact with a database using Zend_Db and Zend_Db_Table
- Secure your application using Zend_Acl and Zend_Auth
- Optimize your application using caching and other performance tips
- Add administrative functionality to your applications

Critique du livre par Fabrice Pestre

Autant le dire dès le début, ce livre n'est pas pour les débutants en POO ou/et avec Zend Framework. Il faudra maîtriser les concepts de base, et plus encore, avant d'en entreprendre la lecture.

L'application développée dans cet ouvrage est une boutique en ligne, front-office et back-office. D'un point de vue technique, une grande partie des aspects de la programmation objet est utilisée : les interfaces, classes abstraites, etc.

Vous apprendrez donc à maîtriser :

- l'architecture MVC ;
- les bases de données avec Zend_DB et Zend_Db_Table ;
- l'authentification (droits et privilèges).
- l'optimisation en utilisant les systèmes de mise en cache ;
- les formulaires et les décorateurs ;
- ZF_Tool pour la création d'un projet de démarrage ;
- les tests unitaires.

En utilisant de manière poussée les principes de la POO, on se perd rapidement dans ce dédale de classes et d'interfaces. Même si la création d'une boutique en ligne, de A à Z, requiert beaucoup de rigueur, de temps et de réflexion, Zend Framework nous permet néanmoins de développer cela bien plus simplement. Malgré tout, cet ouvrage reste très complet, didactique et bien conçu. Les sources sont téléchargeables chapitre par chapitre.

Pour conclure, c'est un très bon ouvrage réservé à un public averti. Vous y trouvez sûrement une foule d'informations à mettre en pratique dans vos développements.

Retrouvez cette critique de livre sur la page livres PHP : Lien18

Les sélecteurs en CSS3

Les sélecteurs permettent, par le biais d'une "requête CSS", d'atteindre un ensemble de nœuds dans un document HTML et de leur donner un style.
Ces "requêtes" sont des règles de reconnaissance de motifs qui déterminent les règles de style qui s'appliquent aux éléments du DOM.
En CSS3, il y a donc des nouveautés au niveau des sélecteurs afin d'atteindre des nœuds dans le DOM de manière encore plus précise.

1. Compatibilité

Chrome, Safari, Opera, Firefox 4, Internet Explorer 9.

2. Rappel

Voici un tableau sur la syntaxe des sélecteurs en CSS2 (référence W3C traduit sur yoyodesign : Les sélecteurs (Lien19)) :

Motif	Signification
*	Correspond à tout élément.
E	Correspond à tout élément E (c.à.d., un élément de type E).
E F	Correspond à tout élément F qui est un descendant de l'élément E.
E > F	Correspond à tout élément F aussi un enfant de l'élément E.
E:first-child	Correspond à un élément E aussi le premier enfant de son élément parent.
E:link, E:visited	Correspond à un élément E qui est une ancre dans la source dont le lien n'a pas été visité (:link) ou bien l'a déjà été (:visited).
E:active, E:hover, E:focus	Correspond à l'élément E au cours de certaines actions de l'utilisateur.
E:lang(c)	Correspond à l'élément de type E qui emploie une langue c (la détermination de cette langue est spécifique au langage du document).
E + F	Correspond à tout élément F immédiatement précédé par un élément E.
E[foo]	Correspond à tout élément E avec l'attribut "foo" (quelles qu'en soient les valeurs).
E[foo="warning"]	Correspond à tout élément E dont l'attribut "foo" a exactement la valeur "warning".
E[foo~="warning"]	Correspond à tout élément E dont l'attribut "foo" a pour valeur une liste de valeurs séparées par des caractères blancs et dont une de celles-ci est "warning".
E[lang|="en"]	Correspond à tout élément E dont l'attribut "lang" a pour valeur une liste de valeurs séparées par des tirets, cette liste commençant (à gauche) par "en".
DIV.warning	Seulement en HTML. Identique à DIV[class~="warning"].
E#myid	Correspond à tout élément E dont l'ID est "myid".

3. Les nouveautés CSS3

Les nouveautés incluent entre autres les **sélecteurs d'attributs**, le **combinateur d'adjacence directe**, les **pseudo-classes** et les **pseudo-éléments**.

4. Les sélecteurs d'attributs

Il y a 3 nouveaux sélecteurs

[attr^="stringValue"]

Ce sélecteur permet de sélectionner un élément DOM dont l'attribut "attr" commence exactement par la valeur "stringValue".

Exemple :

Code CSS :

```
p.example{
  margin:0;
  padding:10px;
  color:#000;
}
p.example[title^="ess"]{
  color:#fff;
  background:#333;
}
```

Code (X)HTML :

```
<p class="example"> je n'ai pas d'attribut title</p>
<p class="example" title="comment"> j'ai un attribut title mais il ne commence pas par "ess"</p>
<p class="example" title="essai"> j'ai un attribut title commençant par "ess"</p>
<p class="example" title="esson"> j'ai un attribut title commençant par "ess" également</p>
```

Démo (Lien20)

[attr$="stringValue"]

Ce sélecteur permet de sélectionner un élément DOM dont l'attribut "attr" finit exactement par la valeur "stringValue".

Exemple :

Code CSS :
```
p.example2{
  margin:0;
  padding:10px;
  color:#000;
}
p.example2[title$="sai"]{
  color:#fff;
  background:#045FB4;
}
```

Code (X)HTML :
```
<p class="example2"> je n'ai pas d'attribut title</p>
<p class="example2" title="comment"> j'ai un attribut title mais il ne commence pas par "ess"</p>
<p class="example2" title="essai"> j'ai un attribut title commençant par "ess"</p>
<p class="example2" title="esson"> j'ai un attribut title commençant par "ess" également</p>
```

Démo (Lien21)

[attr*="stringValue"]

Ce sélecteur permet de sélectionner un élément DOM dont l'attribut "attr" comporte au moins une fois la valeur "stringValue".

Exemple :

Code CSS :
```
p.example3{
  margin:0;
  padding:10px;
  color:#000;
}
p.example3[title*="val"]{
  color:#fff;
  background:#990000;
}
```

Code (X)HTML :
```
<p class="example3"> je n'ai pas d'attribut title</p>
<p class="example3" title="comment"> j'ai un attribut title mais il ne contient pas "val"</p>
<p class="example3" title="val"> j'ai un attribut title contenant au moins "val"</p>
<p class="example3" title="evaluer"> j'ai un attribut title contenant au moins "val" également</p>
<p class="example3" title="eval"> j'ai un attribut title contenant au moins "val" également</p>
```

Démo (Lien22)

5. Le combinateur d'adjacence directe

Permet d'ajouter un style à tous les éléments qui suivent un élément particulier.

Exemple :

Code CSS :
```
.example4 div{
  margin:0;
  padding:10px;
  color:#000;
}
.example4 div~p{
  color:#fff;
  margin:20px;
  width:200px;
  padding:5px;
  border:1px solid #333;
  background:#006644;
}
```

Code (X)HTML :
```
<div class="example4">

  <div>je suis l'élément particulier div</div>
  <p> je suis un p qui suit le div (l'élément particulier)</p>
  <p>je suis un p qui suit le div (l'élément particulier)</p>
  <span>je suis un span</p>
  <p>je suis un p qui ne suit pas le div (l'élément particulier)</p>

</div>
```

Démo (Lien23)

6. Les pseudo-classes

Plusieurs pseudo-classes ont été ajoutées :

:root

Ce sélecteur représente un élément qui est la racine d'un document. Par exemple, en HTML 4, l'élément est *html*.

:nth-child(*expression*)

Ce sélecteur permet de cibler tous les éléments en se basant sur leur position dans la liste des enfants de leur parent.

expression peut être un nombre, une expression numérique ou un mot clé tel que "odd" ou "even".

Exemple :

Code CSS :
```
.exampleTable{
  width:100%;
  border:1px solid #444;
}
/* tous les enfants aux numéros pairs */
.exampleTable tr:nth-child(even){
  background:#999999;
  text-shadow: 2px 2px 5px #111;
  color:#fff;
}
```

```
/* tous les enfants aux numéros impairs */
.exampleTable tr:nth-child(odd){
  background:#990000;
  color:#fff;
}
/* tous les 3 enfants */
.exampleTable tr:nth-child(3n){
  background:#045FB4;
  color:#fff;
}
/* l'enfant numéro 7 */
.exampleTable tr:nth-child(7){
  background:#006400;
  text-shadow: 2px 2px 2px #fff;
  color:#000;
}
```

Code (X)HTML :

```
<table class="exampleTable">
  <tr>
    <td>1ere ligne</td>
  </tr>
  <tr>
    <td>2eme ligne</td>
  </tr>
  <tr>
    <td>3eme ligne</td>
  </tr>
  <tr>
    <td>4eme ligne</td>
  </tr>
  <tr>
    <td>5ere ligne</td>
  </tr>
  <tr>
    <td>6eme ligne</td>
  </tr>
  <tr>
    <td>7eme ligne</td>
  </tr>
  <tr>
    <td>8eme ligne</td>
  </tr>
</table>
```

Démo (Lien24)

:nth-last-child(*expression*)

Ce sélecteur accepte les mêmes arguments que :nth-child() et correspond au dernier enfant d'un élément parent.

C'est le même principe que le :nth-child() sauf que l'on part de la fin.

Exemple :

Code CSS :

```
.exampleTable2{
  width:100%;
  border:1px solid #444;
}
/*tous les enfants aux numéros impairs depuis la fin.*/
.exampleTable2 tr:nth-last-child(odd){
  background:#990000;
  color:#fff;
}
/*les 2 derniers enfants.*/
.exampleTable2 tr:nth-last-child(-n+2){
  background:#045FB4;
  color:#fff;
}
/*l'enfant numéro 7 en partant de la fin donc la 2e ligne du tableau.*/
.exampleTable2 tr:nth-last-child(7){
  background:#006400;
  text-shadow: 2px 2px 2px #fff;
  color:#000;
}
```

Code (X)HTML :

```
<table class="exampleTable2">
  <tr>
    <td>1ere ligne</td>
  </tr>
  <tr>
    <td>2eme ligne</td>
  </tr>
  <tr>
    <td>3eme ligne</td>
  </tr>
  <tr>
    <td>4eme ligne</td>
  </tr>
  <tr>
    <td>5ere ligne</td>
  </tr>
  <tr>
    <td>6eme ligne</td>
  </tr>
  <tr>
    <td>7eme ligne</td>
  </tr>
  <tr>
    <td>8eme ligne</td>
  </tr>
</table>
```

Démo (Lien25)

:last-child

Correspond à ***:nth-last-child(1)*** .

:first-child

Correspond à ***:nth-child(1)*** .

:nth-of-type(*expression*)

Ce sélecteur représente un élément qui a *expression* frères du même type **devant** lui dans l'arbre DOM.

Code CSS :

```
img:nth-of-type(2n+1) { float: right; }
img:nth-of-type(2n) { float: left; }
```

alternance de la position des images en flottant.

:nth-last-of-type(*expression*)

Ce sélecteur représente un élément qui a *expression - 1* frères du même type après lui dans l'arbre DOM.

```
body > h2:nth-of-type(n+2):nth-last-of-type(n+2)
```

représente tous les h2 fils d'un élément XHTML body sauf le premier et le dernier.

:first-of-type

Correspond à :nth-of-type(1). :first-of-type représente un élément qui est le **premier** enfant de son type dans la liste des enfants de son élément parent.

:last-of-type

Correspond à :nth-last-of-type(1). Représente un élément qui est le **dernier** enfant de son type dans la liste des enfants de son élément parent.

:only-child

Correspond à un élément qui n'a **aucun frère**. Cette pseudo-classe correspond à ***:first-of-type:last-of-type*** ou ***:nth-of-type(1):nth-last-of-type(1)*** .

:checked

Correspond aux éléments cochés d'un formulaire.

:contains(*value*)

Correspond aux éléments dont le contenu textuel contient la sous-chaîne donnée en argument.

Exemple :

Code CSS :
```
p:contains('essai'){
  background:#900;
}
```

signifie que tous les éléments "p" contenant la sous-chaîne "essai" auront pour couleur d'arrière plan, la valeur '#900'.

L'usage de la pseudo-classe de contenu (:contain) est restreint aux médias statiques.

:empty

Correspond aux éléments n'ayant pas d'enfant.

:not([*expression*])

Représente un élement qui n'est pas représenté par l'expression donnée en argument.

Exemple:

Code CSS :
```
button:not([DISABLED]){
  ...
}
```

```
a:not(:visited){
  ...
}
```

:target

Représente un élément qui est la cible de l'URI.

Exemple :

Code CSS :
```
p:target { background:#900}
```

Tout élément p qui sera la cible de l'URI (via l'ID # en tant qu'ancre) aura pour couleur d'arrière plan la valeur "#900".

7. Les pseudo-éléments

::first-line

Applique la règle de style à la premiere ligne du texte de l'élément.

Code CSS :
```
p::first-line { text-transform: uppercase }
```

La 1re ligne du ou des éléments "p" est mise en majuscules.

::first-letter

Applique la règle de style à la première lettre du texte de l'élément.

Code CSS :
```
p::first-letter { font-size: 2em }
```

La 1re lettre du ou des éléments "p" a une taille de police de 2em.

::selection

Applique la règle de style à la sélection du texte de l'élément faite par l'utilisateur.

Code CSS :
```
p::selection { background:#006644 }
```

À la sélection, le texte sélectionné aura une couleur d'arrière-plan de valeur '#006644'.

::before et ::after

Génère un contenu avant ou après un contenu d'un élément.

Ces pseudo-éléments existent en CSS2.1 sous forme **:before** et **:after**.

Retrouvez l'article de Jérôme Debray en ligne : Lien26

Comprendre les couleurs en CSS3

Cet article a pour but de présenter la nouvelle gestion des couleurs en CSS3.

1. Compatibilité

Chrome, Safari, Opera, Firefox 4, Internet Explorer 9.

2. Couleur HSL

Le CSS3 inclut maintenant dans la norme, la notion de **teinte**, de **saturation** et de **luminosité** (**H**ue, **S**aturation, **L**uminosity).

Le **Hue** permet de modifier la teinte donc la **couleur**, c'est une valeur numérique.

Par exemple, 0 (ou 360) représente le rouge, 120 le vert et 240 le bleu ; c'est une mesure sur la roue chromatique. Entre ces trois valeurs, ce sont les nuances.

- entre 0 et 119 (nuance de rouge) ;
- entre 120 et 239 (nuance de vert) ;
- entre 240 et 359 (nuance de bleu).

La **saturation** est une valeur en pourcentage, 100% équivalent à la couleur.

La **luminosité** est une valeur en pourcentage :

- 0% est le sombre (noir) ;
- 50% est la moyenne ;
- 100% est la lumière (blanc).

Exemple :

CSS

```
p.hsl{
  background-color:hsl(100, 100%, 50%);
  color:hsl(250, 100%, 50%);
}
p.hslAutre{
  background-color:hsl(100, 100%, 50%);
  /* couleur plus sombre que la précédente */
  color:hsl(250, 100%, 25%);
}
```

HTML

```
<p class="hsl"> couleurs gérées en hsl </p>
<p class="hslAutre"> couleurs gérées en hsl </p>
```

Démo (Lien27)

3. Couleur HSLA

C'est la même chose que la couleur **hsl**, sauf qu'il y a en plus la **gestion de l'opacité** de la couleur (sa transparence).

Exemple :

CSS

```
p.hsla{
  background-color:hsl(100, 100%, 50%, 0.8);
  color:hsla(250, 100%, 50%, 0.5);
}
p.hslaAutre{
  background-color:hsla(90, 100%, 50%, 0.5);
  color:hsla(250, 100%, 50%, 1.0);
}
```

HTML

```
<p class="hsla"> couleurs gérées en hsla </p>
<p class="hslaAutre"> couleurs gérées en hsla
</p>
```

Démo (Lien28)

4. Couleur RGBA

Le **RGB** existe depuis un moment dans la norme CSS. Avec le CSS3, l'opacité est appliquée au **RGB** via **RGBA**.

Les valeurs sont les suivantes :

- R : couleur rouge en % (de 0% à 100%) ou en valeur (0 à 255) ;
- G : couleur verte en % (de 0% à 100%) ou en valeur (0 à 255) ;
- B : couleur bleue en % (de 0% à 100%) ou en valeur (0 à 255) ;
- A : opacité en valeur flottante (0.0 à 1.0).

Exemple :

CSS

```
p.rgba{
  background-color:rgba(255, 0, 0, 0.2);
  color:rgb(250, 100, 50);
}
p.rgbaAutre{
  background-color:rgba(90, 100, 50, 0.5);
  color:rgba(250, 100, 50, 0.4);
}
```

HTML

```
<p class="rgba"> couleurs gérées en rgba </p>
<p class="rgbaAutre"> couleurs gérées en rgba
</p>
```

Démo (Lien29)

L'intérêt du **RGBA** et du **HSLA** vient du fait que l'opacité qu'ils appliquent aux éléments, **n'est pas héritée par les enfants de ces éléments** (c'est le cas pour **opacity**).

5. Opacity

L'**opacity** permet d'appliquer une transparence **à un élément et à tous ses enfants** (contrairement à **HSLA** et à **RGBA**)

Il prend pour valeur un flottant (entre 0.0 et 1.0)

Exemple:

CSS

```
p.opacity{
  opacity:0.5;
}
```

```
p.opacityAutre{
  opacity:0.3;
}
```

HTML

```
<p class="opacity"> exemple sur opacity </p>
<p class="opacityAutre"> exemple sur opacity </p>
```

Démo (Lien30)

Retrouvez l'article de Jérôme Debray en ligne : Lien31

Coins arrondis CSS

Pour réaliser des blocs (menus ou autres) avec des coins arrondis, il existe principalement trois méthodes en CSS, qui offrent plus ou moins de fluidité et de possibilités graphiques.
Créer des coins arrondis avec une seule image, coulissante en hauteur et en largeur, permet une certaine liberté graphique et s'adapte à un design fluide.

1. Préambule

Les trois méthodes courantes pour créer des coins arrondis sont :

1. La propriété CSS 3 border-radius (implémentation très attendue !) ;
2. La technique positionnant quatre images de coins à l'aide de quatre div imbriqués ;
3. La technique de l'image coulissante en hauteur.

Chacune de ces méthodes a ses avantages et ses inconvénients.

Cette 4e méthode, utilisant une seule image adaptable en hauteur et en largeur pour un design fluide, s'inspire des deux dernières pré citées. Elle souffre aussi de quelques défauts, notamment l'éternelle imbrication de plusieurs div. Cependant, elle a l'avantage d'offrir plus de possibilités graphiques (et donc pour faire autre chose que des coins arrondis) que la seconde technique. Attention cependant à ce que cette image de fond ne soit pas trop lourde !

Fonctionne avec :

- tous les navigateurs graphiques.

Attributs utilisés :

- background ;
- margin ;
- max-width (inopérant avec IE6) ;
- padding.

2. Image de fond

Le but est donc de mettre en fond une seule image, très grande, qui pourra s'adapter aux résolutions les plus courantes. La taille des écrans s'agrandissant, j'ai tablé sur une largeur d'environ 2000 pixels, la hauteur en faisant autant. Bien entendu, si vous pensez que vos pages pourront dépasser 2000 pixels de hauteur, il faudra la rallonger d'autant !

L'image a un poids d'un peu moins de 17 ko, elle est donc très légère et se chargera très vite.

2.1. Code (X)HTML

Il faut donc quatre div (eh oui...) pour permettre de positionner correctement l'image autour du texte.

```
<div id="gauche">
  <div id="droite">
    <div id="haut">
      <div></div>
    </div><!-- /haut -->
    <h2>Lorem ipsum</h2>
    <p>
      Lorem ipsum dolor sit amet, consectetur
adipiscing elit.
      Mauris vulputate laoreet urna. Integer
magna.
      Donec facilisis lectus sed quam.
      Curabitur sit amet lacus id lacus facilisis
venenatis.
    </p>
  </div><!-- /droite -->
</div><!-- /gauche -->
```

2.2. Code CSS

On commence par positionner le coin bas gauche de l'image de fond à l'aide de la propriété background. Le margin:auto est facultatif, c'est juste pour centrer le bloc dans la page. Par contre, la précaution du max-width, correspondant à la largeur réelle de l'image, est indispensable pour les résolutions d'écran supérieures à celle-ci : elles sont encore rares, mais suffisantes pour en tenir compte.

Pour rappel, le code :

```
background-color:#E4EFFF;
background-image:url(images/fond-arrondi.png);
background-repeat:no-repeat;
background-position:bottom left;
```

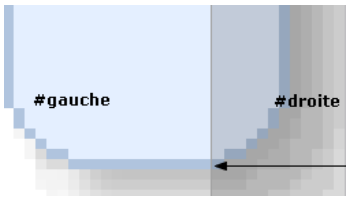
peut s'optimiser par :

```
background:#E4EFFF url(images/fond-arrondi.png)
no-repeat bottom left;
```

Cela donne donc :

```
#gauche {
  background:#E4EFFF url(images/fond-arrondi.png)
no-repeat bottom left;
  margin:auto;
  max-width:2007px;
}
```

On positionne ensuite le bas de l'image à droite (bottom right). C'est donc le premier "coulissement" de l'image en faisant glisser le côté droit sur le côté gauche. Les sept pixels déclarés en marge de gauche (margin-left) correspondent à la largeur de l'arrondi qui ne doit pas être recouvert.




```
#droite {
  background:#E4EFFF url(images/fond-arrondi.png)
no-repeat bottom right;
  margin-left:7px;
  padding-bottom:20px;
}
```

Vient ensuite le tour du positionnement du haut de l'image avec l'élément supérieur du bloc délimité par le et l'autre div imbriqué. C'est le second coulissement, cette fois en hauteur. Les tailles de sept pixels correspondent toujours aux tailles des coins.

```
#haut {
  background:#E4EFFF url(images/fond-arrondi.png)
no-repeat top right;
  margin-left:-7px;
  padding:0;
}
#haut div {
  background:#E4EFFF url(images/fond-arrondi.png)
no-repeat top left;
  height:7px;
  width:7px;
}
```

Ne reste plus qu'à définir les marges désirées pour que le texte ne colle pas aux bords de l'image sans interférer avec les positionnements précédents.

```
p, h2 {
  padding:5px 10px;
  margin:0;
}
```

Voir le résultat (Lien32)

3. Lectures complémentaires

- Coins arrondis en CSS de Maurice Svay : Lien33 ;
- Diverses méthodes pour créer des coins arrondis (Biboo) : Lien34 ;
- Coins arrondis pour designs fluides : Lien35 ;
- CSS3 info : la propriété border-radius (en) : Lien36 ;
- Générateur de coins arrondis (en) : Lien37.

Retrouvez l'article de Pascale Lambert en ligne : Lien38

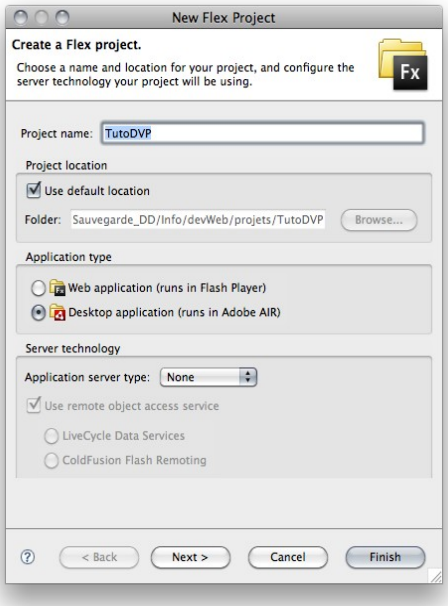
Créer une application multiplateforme AIR utilisant SQLite

Dans ce tutoriel nous allons créer avec Flex Builder 3 un exemple d'application AIR utilisant SQLite.
Cette application doit permettre de gérer une liste de contacts stockée dans une base de données.
Consultez aussi l'article sur l'utilisation de SQLite dans AIR : Lien39.

1. Création d'un nouveau projet AIR

La première chose à faire est d'initialiser un nouveau projet AIR en utilisant le menu suivant :
File > New > Flex Project

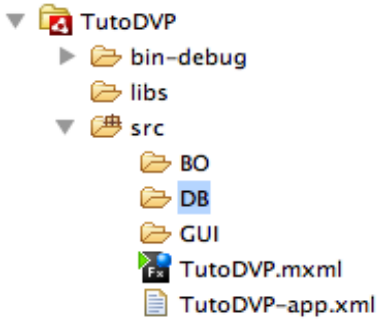
Sélectionnez ensuite l'option application de bureau AIR (Desktop application) puis validez.



Le dossier source de l'application créée contient le fichier mxml avec la définition de l'interface principale de l'application :

Interface par défaut
```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical">
</mx:WindowedApplication>
```

Le projet sera organisé avec les trois packages suivants : **BO** pour les objets métiers (ici une seule classe Contact), **DB** pour les classes d'accès aux données de la base et **GUI** qui contiendra les interfaces de l'application.



2. Définition de l'objet Contact

Pour commencer il faut définir notre contact. Les propriétés d'un contact sont un pseudonyme, une date de naissance, une adresse mail et un avatar.
Pour stocker l'avatar on utilise une variable de type ByteArray. La propriété ID correspondra à un identifiant auto-incrémenté dans la base de données.

classe Contact
```
package BO
{
  import flash.utils.ByteArray;

  public class Contact
  {
    public var id:int;
    public var pseudo:String;
    public var dateNaissance:Date;
    public var mail:String;
    public var avatar:ByteArray;

    public function Contact()
    {
      ID = -1;
      Pseudo = "";
      Mail = "";
      DateNaissance = null;
      Avatar = null;
    }

    public function get ID():int
    {
      return id;
    }
    public function set ID(pValue:int):void
    {
      id = pValue;
    }

    public function get Pseudo():String
    {
      return pseudo;
```

```
    }
    public function set
Pseudo(pValue:String):void
    {
      pseudo = pValue;
    }

    public function get Mail():String
    {
      return mail;
    }
    public function set Mail(pValue:String):void
    {
      mail = pValue;
    }

    public function get Avatar():ByteArray
    {
      return avatar;
    }
    public function set
Avatar(pValue:ByteArray):void
    {
      avatar = pValue;
    }

    public function get DateNaissance():Date
    {
      return dateNaissance;
    }
    public function set
DateNaissance(pDate:Date):void
    {
      dateNaissance = pDate;
    }
  }
}
```

3. Initialisation des classes d'accès aux données

Il faut définir une interface d'accès à notre base de contacts qui fournira les fonctionnalités suivantes : la sélection des contacts de la base, l'insertion, la mise à jour et la suppression de contacts.

Nous allons créer une classe Database dans le package DB pour effectuer l'exécution des requêtes.
Nous devrons donc gérer au sein de cette classe le fichier de la base, la connexion à la base, l'initialisation et l'exécution des objets SQLStatement.

Il est aussi nécessaire de gérer les erreurs lors de l'exécution des requêtes et de permettre de récupérer la description d'une éventuelle erreur.
Afin d'alléger un peu le code, la classe Database héritera d'une classe Logger, qui permettra de stocker et de mettre à disposition la description des erreurs survenues.

La classe *Logger* :

Classe Logger

```
package DB
{
  import flash.events.EventDispatcher;

  public class Logger extends EventDispatcher
  {
    private var errorMessageList:Array;

    public function get LastErrorMessage():String
    {
      return errorMessageList.length > 0 ?
errorMessageList[errorMessageList.length-1] : "No
error";
    }

    public function Logger()
    {
      errorMessageList = new Array();
    }

    protected function
logError(pErrorMessage:String):void
    {
      errorMessageList.push(pErrorMessage);
      trace(pErrorMessage);
    }
  }
}
```

La classe Database est uniquement initialisée (voir ci-dessous), les différentes méthodes nécessaires seront rajoutées au moment voulu.

La classe Database possède donc une référence, vers le fichier de la base, qui est initialisée dans le constructeur et qui ne changera pas.

classe Database

```
package DB
{
  import flash.filesystem.File;
  import flash.data.SQLConnection;
  import flash.data.SQLStatement;

  public class Database extends Logger
  {
    private var databaseFile:File;

    public function Database()
    {
      databaseFile =
File.applicationStorageDirectory.resolvePath("dat
abase.sqlite");
    }
  }
}
```

Pour pouvoir accéder facilement à la base de contacts dans le reste de l'application, la classe Database sera un Singleton (une instance unique accessible de façon statique).

L'instance est accessible avec la propriété ***Database.Instance***.

classe Database

```
package DB
{
  import flash.filesystem.File;

  public class Database extends Logger
  {
    private static var constructorKey:Object =
{};
```

```
    private static var instance:Database = null;

    private var databaseFile:File;

    public function
Database(pConstructorKey:Object)
    {
      if(pConstructorKey != constructorKey)
      {
        throw new Error("Instanciation illégale
(constructeur privé)");
      }

      databaseFile =
File.applicationStorageDirectory.resolvePath("dat
abase.sqlite");
    }

    public static function get
Instance():Database
    {
      if(instance == null)
      {
        instance = new Database(constructorKey);
      }
      return instance;
    }
  }
}
```

4. Initialisation de la base de données

Lors de l'initialisation de l'application il faut vérifier si la base existe et la créer dans le cas contraire. Nous créons donc une méthode ***createDatabase*** qui retourne un booléen pour indiquer si la création s'est bien déroulée.

Méthode de création de la base

```
public function createDatabase():Boolean
{
  if(databaseFile.exists)
  {
    return true;
  }
  // la suite ici
}
```

Premièrement il faut construire la requête de création de la table Contact. Il faut s'assurer que les noms des champs de la table correspondent aux noms des propriétés de l'objet Contact créé précédemment. La requête est la suivante :

Requête de création

```
CREATE TABLE Contact (ID INTEGER PRIMARY KEY,
Pseudo TEXT, Mail TEXT, DateNaissance Date,
Avatar OBJECT)
```

Nous devons maintenant créer une méthode pour exécuter la requête, méthode qui sera privée car uniquement à utiliser dans cette classe. Elle assure l'ouverture et la fermeture de la connexion, la définition d'une variable SQLStatement et son exécution, puis retourne le résultat de la requête. Si une erreur se produit, la valeur null est retournée.

Méthode pour l'exécution des requêtes

```
private function executeQuery(pMode:SQLMode,
pQuery:String):SQLResult
{
  try
  {
      var Connection:SQLConnection = new
SQLConnection();
      var Statement:SQLStatement = new
SQLStatement();

      Connection.open(databaseFile, pMode);

      Statement.sqlConnection = Connection;

      Statement.text = pQuery;
      Statement.execute();

      return Statement.getResult();
  }
  catch(error:SQLError)
  {
      this.logError("Requête : <" + pQuery +
">\n" + error.toString());
  }
  finally
  {
      Connection.close();
  }
    return null;
}
```

La méthode de création est donc la suivante :

Méthode de création de la base

```
public function createDatabase():Boolean
{
  if(databaseFile.exists)
  {
    return true;
  }

  var sQuery:String = "CREATE TABLE Contact (";

  sQuery += "ID INTEGER PRIMARY KEY, "
  sQuery += "Pseudo TEXT, "
  sQuery += "Mail TEXT, "
  sQuery += "DateNaissance Date, "
  sQuery += "Avatar OBJECT "
  sQuery += ")";

  var result:SQLResult =
this.executeCreateQuery(SQLMode.CREATE, sQuery);

  return result != null;
}
```

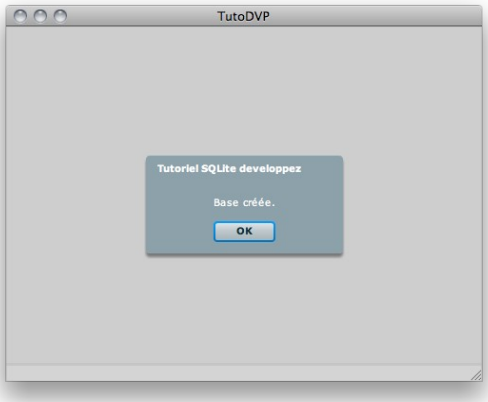
Il reste à appeler cette méthode lors de l'initialisation de l'interface principale, pour cela il faut définir l'évènement creationComplete de l'application dans le fichier principal (TutoDVP.mxml) :

Interface principale

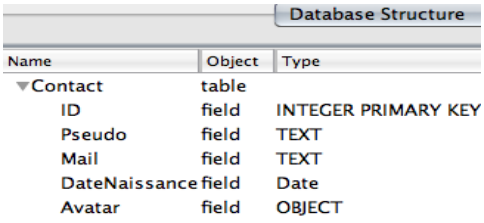
```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication
xmlns:mx="http://www.adobe.com/2006/mxml"
layout="vertical"
creationComplete="InitApplication()"
                          width="500" height="470"
```

```
title="Tutoriel AIR SQLite developpez"
viewSourceURL="srcview/index.html">
  <mx:Script>&lt;![CDATA[
      import mx.controls.Alert;
      import DB.Database;

      private function initApplication():void
      {
        if(!Database.Instance.createDatabase())
        {
          Alert.show("Erreur lors de la création
de la base :\n" +
Database.Instance.LastErrorMessage, "Tutoriel
SQLite developpez");
        }else{
          Alert.show("Base créée.", "Tutoriel
SQLite developpez");
        }
      }

  ]]&gt;</mx:Script>
</mx:WindowedApplication>
```

Vous pouvez déjà lancer l'application pour créer la base !

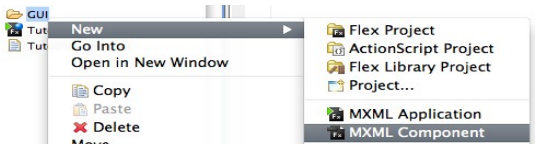


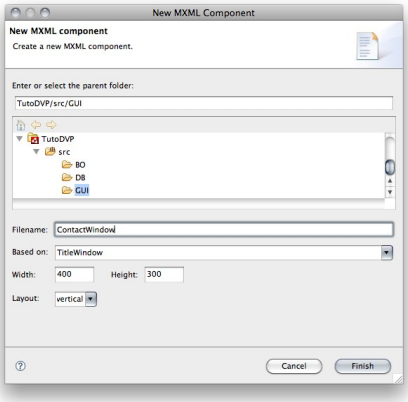
On peut ensuite vérifier que la table est bien créée à l'aide d'un utilitaire qui permet d'explorer les bases SQLite :



5. Création de l'interface pour l'ajout d'un contact

Maintenant que notre base de données est créée il faut y insérer des données. Nous allons créer une fenêtre contenant un formulaire pour la création d'un nouveau contact. Cette fenêtre sera un composant basé sur un objet TitleWindow. Pour le créer, sélectionnez le menu correspondant, puis donnez un nom et une classe de base comme ci-après :





Cette fenêtre va contenir un formulaire comprenant trois champs de saisie pour le pseudonyme, l'adresse mail et la date de naissance, puis une image avec un bouton pour sélectionner l'avatar.
Lorsque le formulaire est rempli et validé, un objet Contact est créé à partir des différents éléments.

Voir le code en ligne : Lien40

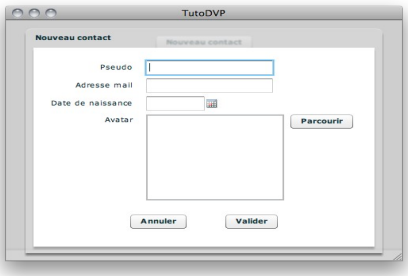
Dans l'interface principale on ajoute un bouton "Nouveau contact" qui va ouvrir la fenêtre de création de contacts :

```
<mx:HBox>
   <mx:Spacer width="100%" />
   <mx:Button label="Nouveau contact"
horizontalCenter="true" click="addContact()" />
   <mx:Spacer width="100%" />
</mx:HBox>
```

avec la fonction addContact qui utilise la classe PopUpManager :

```
private function addContact():void
{
   var window:IFlexDisplayObject =
PopUpManager.createPopUp(this, ContactWindow ,
true);
   var CreationWindow:ContactWindow =
ContactWindow(window);
   PopUpManager.centerPopUp(CreationWindow);
}
```

L'interface est maintenant créée, lancez l'application pour vérifier que la fenêtre réagit bien.



Bien sûr, il faut maintenant créer la fonction d'insertion du contact dans la base de données.

6. Insérer un nouveau contact

Pour l'insertion du nouveau contact nous allons créer la méthode correspondante dans la classe Database. Comme pour la création de la base il faut d'abord construire la requête puis l'exécuter. Nous devons en plus récupérer le nouvel identifiant automatique généré.

L'exécution de cette requête nécessite la définition de paramètres, il faut donc enrichir la méthode executeQuery avec un nouveau paramètre pParameters de type Object. Ce paramètre aura la valeur null par défaut pour ne pas modifier le comportement initial de la méthode.

Méthode pour l'exécution des requêtes

```
private function executeQuery(pMode:String,
pQuery:String, pParameters:Object =
null):SQLResult
{
   try
   {
       var Connection:SQLConnection = new
SQLConnection();
       var Statement:SQLStatement = new
SQLStatement();

       Connection.open(databaseFile, pMode);

       Statement.sqlConnection = Connection;

       Statement.text = pQuery;

       if(pParameters != null)
       {
           for(var id:String in pParameters)
           {
               Statement.parameters[id] =
pParameters[id];
           }
       }

       Statement.execute();
       return Statement.getResult();
   }
   catch(error:SQLError)
   {
       this.logError("Requête : <" + pQuery +
">\n" + error.toString());
   }
   finally
   {
       Connection.close();
   }
   return null;
}
```

La requête à exécuter est la suivante, les paramètres sont préfixés par le caractère @ :

Requête d'insertion

```
INSERT INTO Contact (Pseudo, Mail, DateNaissance,
Avatar) VALUES (@Pseudo, @Mail, @DateNaissance,
@Avatar)
```

La méthode insertContact lance l'exécution en spécifiant la requête ainsi que les paramètres puis récupère le nouvel identifiant du contact :

Méthode pour l'insertion d'un contact dans la base

```
public function
insertContact(pContact:Contact):Boolean
{
   var sQuery:String = "INSERT INTO Contact
(Pseudo, Mail, DateNaissance, Avatar) VALUES
(@Pseudo, @Mail, @DateNaissance, @Avatar)";
   var params:Object = {"@Pseudo" :
pContact.Pseudo, "@Mail" : pContact.Mail,
"@DateNaissance" : pContact.DateNaissance,
"@Avatar" : pContact.Avatar};

   var result:SQLResult =
this.executeQuery(SQLMode.UPDATE, sQuery,
params);

   if(result != null)
   {
       pContact.ID = result.lastInsertRowID;
       return true;
   }

   return false;
}
```

Retournons dans la fenêtre de création pour appeler la méthode insertContact :

```
var co:Contact = new Contact();
co.Pseudo = inputPseudo.text;
co.Mail = inputMail.text;
co.DateNaissance = inputDate.selectedDate;
co.Avatar = imageToByteArray(inputAvatar);

if(Database.Instance.insertContact(co))
{
   Close();
}
else
{
   Alert.show("Erreur lors de
l'enregistrement\n\n" +
Database.Instance.LastErrorMessage, "Tutoriel
SQLite developpez");
}
```

Lancez l'application, l'interface est fonctionnelle. Vous pouvez maintenant ajouter des contacts dans la base.

Retrouvez la suite de l'article de Julien Fray en ligne : Lien40

Utiliser SQLite dans une application AIR

Cet article explique comment utiliser les différentes fonctionnalités du moteur d'exécution SQLite embarqué dans AIR.
Consultez aussi le tutoriel et créez une application multiplateforme AIR avec SQLite : Lien40.

1. Introduction

L'environnement AIR (Adobe Integrated Runtime) permet de créer des applications riches exécutées en local sous la forme d'applications clientes autonomes sous toutes les plateformes.

Il y a de multiples raisons d'utiliser SQLite avec AIR :

- SQLite est open source et implémente la majeure partie de la norme SQL-92 ;
- La base de données est contenue dans un fichier unique et léger, aucun serveur n'est donc nécessaire ;
- SQLite est une bibliothèque écrite en C qui est embarquée dans Adobe AIR, il n'y a donc aucune librairie supplémentaire à utiliser.

AIR permet l'utilisation de SQLite en mode synchrone ou asynchrone.
L'avantage d'utiliser le mode asynchrone est que l'interface n'est pas figée pendant les opérations mais l'inconvénient est que le nombre de lignes de code est plus important.
En utilisant le mode synchrone le code est plus simple et cela reste suffisant dans la plupart des cas.

2. Connexion

La base de données étant stockée dans un fichier, unique il faut donc déterminer le fichier à utiliser pour établir la connexion à la base.

Dans l'exemple, l'objet de type File pointe sur le fichier "database.sqlite", présent dans le répertoire de stockage de l'application.
Ce répertoire unique et spécifique à l'application est créé lors du premier accès à la propriété applicationStorageDirectory.
Il utilise le protocole app-storage: et non le classique file:, l'URL du fichier dans l'exemple est donc "app-storage:/database.sqlite".
L'emplacement du dossier sur le disque peut varier suivant le système d'exploitation. Un exemple sur MacOS : /Users/<nom utilisateur>/Library/Preferences/<nom application>/Local Store.

Lorsque la base de données n'a pas déjà été créée, l'objet File doit pointer sur un fichier inexistant. Cela peut être vérifié avec la propriété File.exists et dans ce cas il faut utiliser un mode de connexion spécifique à savoir SQLMode.CREATE pour que le fichier soit créé lors de la connexion.

Les différents modes de connexion sont les suivants :

- SQLMode.READ : la connexion est ouverte en mode lecture seule ;
- SQLMode.UPDATE : la connexion est ouverte pour les mises à jour mais une nouvelle base de données n'est pas créée si le fichier spécifié n'existe pas ;
- SQLMode.CREATE : la connexion est ouverte pour les mises à jour et un fichier de base de données est créé si le fichier spécifié n'existe pas.

Les méthodes ***open*** et ***openAsync*** de la classe SQLConnection sont utilisées pour établir la connexion.

Exemple synchrone

```
var DatabaseFile:File =
File.applicationStorageDirectory.resolvePath("dat
abase.sqlite");

var connection:SQLConnection = new
SQLConnection();
connection.open(DatabaseFile, SQLMode.READ);
...
connection.close();
```

Exemple asynchrone

```
var DatabaseFile:File =
File.applicationStorageDirectory.resolvePath("dat
abase.sqlite");

var connection:SQLConnection = new
SQLConnection();
connection.addEventListener(SQLEvent.OPEN,
DatabaseOpenSuccessHandler);
connection.openAsync(DatabaseFile, SQLMode.READ);

private function
DatabaseOpenSuccessHandler(evt:SQLEvent):void
{
  ...
  connection.close();
}
```

Il est aussi possible de travailler sur plusieurs bases de données à partir de la même connexion.

La méthode ***attach*** permet d'ajouter une base à la connexion en spécifiant un nom unique pour la base.
De cette façon il est ensuite possible d'indiquer explicitement dans les requêtes la base à laquelle correspond une table en utilisant la syntaxe [nom-base].[nom-table].
Les noms "main" et "temp" ne doivent pas être utilisés.
Comme pour la base principale il faut définir le fichier qui contiendra la base.

L'appel à la méthode ***detach*** supprime la connexion à la base correspondant au nom spécifié.

Exemple synchrone

```
var DatabaseFile:File =
File.applicationStorageDirectory.resolvePath("dat
abase.sqlite");
var DatabaseFile2:File =
File.applicationStorageDirectory.resolvePath("dat
abase2.sqlite");

var connection:SQLConnection = new
SQLConnection();
connection.open(DatabaseFile, SQLMode.READ);
...
connection.attach("autrebase", DatabaseFile2);

// Execution de requêtes avec la syntaxe
autrebase.[nom-table]

connection.detach("autrebase");
...
connection.close();
```

3. Exécution des requêtes

L'exécution des requêtes s'effectue avec la classe SQLStatement.
Il faut spécifier la connexion utilisée qui doit être ouverte au préalable et ensuite définir la requête SQL à exécuter simplement par l'appel à la méthode ***execute***.

Le résultat est récupéré dans un objet SQLResult.

Dans le cas d'une requête d'insertion dans une table avec une clé auto-incrémentée la propriété lastInsertRowID contient l'identifiant généré pour le dernier enregistrement inséré.

Dans le cas d'une requête de mise à jour la propriété rowsAffected contient le nombre d'enregistrements affectés par la mise à jour.

Dans le cas d'une requête de sélection la propriété data est un tableau contenant les enregistrements retournés. Si le résultat ne contient aucun enregistrement la propriété data est égale à null.

Par défaut chaque élément du tableau est une variable Object dont les noms des propriétés correspondent aux noms des colonnes des données du jeu de résultats.

Il est possible de spécifier le type des objets correspondant aux résultats d'une requête en utilisant l'attribut ***itemClass*** (de type Class) de la classe SQLStatement.
De cette façon les objets contenus dans le tableau de résultats ne seront plus de type Object mais du type spécifié.
Ces objets sont instanciés automatiquement.

Attention, la classe spécifiée doit avoir un constructeur qui ne requiert aucun paramètre. De plus, pour chaque champ sélectionné, doit correspondre une propriété de la classe.

Dans les exemples suivants les requêtes sont exécutées sur une simple table "Utilisateur" avec la structure suivante :

Utilisateur
Nom : TEXT
Prenom : TEXT
DateInscription : DATE

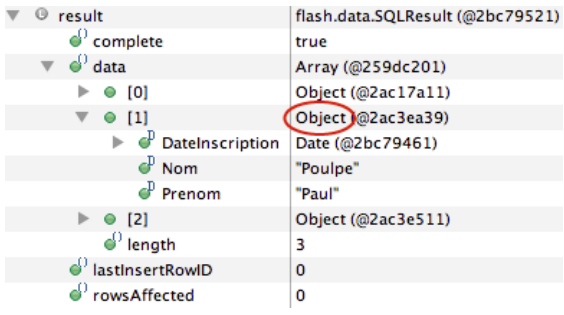
Exemple synchrone

```
var statement:SQLStatement = new SQLStatement();
statement.sqlConnection = connection;
statement.text = "SELECT * FROM Utilisateur";

statement.execute();

var result:SQLResult = statement.getResult();

connection.close();

var count:uint = 0;
if(result.data != null)
{
  count = result.data.length;
}
trace("Il y a " + count + " utilisateur(s)");
```

Exemple asynchrone

```
var statement:SQLStatement = new SQLStatement();
statement.addEventListener(SQLEvent.RESULT,
DatabaseResultHandler);
statement.sqlConnection = connection;
statement.text = "SELECT * FROM Utilisateur";
statement.execute();

private function
DatabaseResultHandler(evt:SQLEvent):void
{
  var result:SQLResult =
SQLStatement(event.target).getResult();

  var count:uint = 0;
  if(result.data != null)
  {
    count = result.data.length;
  }
  trace("Il y a " + count + " utilisateur(s)");
}
```

Par défaut voici les objets récupérés :



L'exemple suivant utilise la classe Utilisateur pour typer les résultats :

Classe Utilisateur

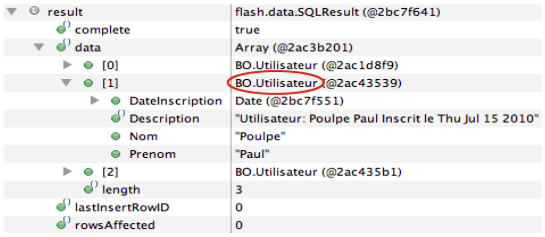
```
public class Utilisateur
{
  public var Nom:String;
  public var Prenom:String;
```

```
  public var DateInscription:Date;

  public function Utilisateur()
  {
    Nom = "";
    Prenom = "";
    DateInscription = new Date();
  }

  public function get Description():String
  {
    return "Utilisateur: " + Nom + " " + Prenom +
" Inscrit le " + DateInscription.toDateString();
  }
}
```

Exemple synchrone avec instanciation automatique

```
var statement:SQLStatement = new SQLStatement();
statement.sqlConnection = connection;
statement.text = "SELECT * FROM Utilisateur";
statement.itemClass = Utilisateur;

statement.execute();

var result:SQLResult = statement.getResult();

connection.close();

Utilisateur ut;
for(var i:uint=0; i<statement.data.length; i++)
{
  ut = statement.data[i];
  trace( ut.Description );
}
```

Les objets Utilisateur récupérés :



4. Paramètres des requêtes

Lors de la construction des requêtes il est préférable d'utiliser les paramètres de l'objet SQLStatement. Cela évite de devoir convertir manuellement les valeurs en chaînes de caractères et donc limite le risque d'erreur.

Les paramètres peuvent être nommés ou non.
Dans le cas de paramètres nommés la propriété parameters est un objet dont le nom des propriétés correspond aux noms des paramètres. Les paramètres doivent être préfixés par le caractère '@' ou ':'.
Si les paramètres ne sont pas nommés (?) la propriété parameters est un tableau contenant l'ensemble des paramètres.

Exemple de paramètres nommés

```
statement.text = "SELECT * FROM Utilisateur WHERE
Name=@Name";
statement.parameters["@name"] = user.name;
```

Exemple de paramètres anonymes

```
statement.text = "SELECT * FROM Utilisateur WHERE
Name=?";
statement.parameters[0] = user.name;
```

5. Gestion des erreurs

La gestion des erreurs lors de la connexion ou de l'exécution d'une requête s'effectue de deux façons différentes, en mode synchrone ou asynchrone.

En mode synchrone il faut gérer les exceptions de type SQLError :

Exemple synchrone

```
try
{
  var statement:SQLStatement = new
SQLStatement();
  statement.sqlConnection = connection;
  statement.text = "SELECT * FROM Utilisateur";

  statement.execute();

  var result:SQLResult = statement.getResult();

  var count:uint = 0;
  if(result.data != null)
  {
    count = result.data.length;
  }
  trace("Il y a " + count + " utilisateur(s)");
}
catch(error:SQLError)
{
  trace("Erreur : " + error.toString());
}
finally
{
  connection.close();
}
```

En mode asynchrone il faut écouter l'évènement SQLErrorEvent.ERROR sur un objet SQLConnection ou SQLStatement :

Exemple asynchrone

```
statement.addEventListener(SQLErrorEvent.ERROR,
DatabaseErrorHandler);

private function
DatabaseErrorHandler(evt:SQLErrorEvent):void
{
  trace("Erreur : " + evt.error.toString());
}
```

6. Transactions

Les transactions sont généralement utilisées pour **assurer l'intégrité des données**.
Elles permettent d'exécuter plusieurs requêtes et de pouvoir revenir à l'état initial de la base dans le cas où une des requêtes provoque une erreur.
Les données contenues dans la base restent donc cohérentes.

Avec SQLite les transactions sont aussi utilisées dans le cas d'une insertion importante de données.

En effet lorsqu'un grand nombre de requêtes d'insertion est effectué à la suite, l'utilisation d'**une transaction améliore grandement les performances**.
L'exécution est jusqu'à deux fois plus rapide pour l'insertion de quelques milliers d'enregistrements.

Les transactions sont gérées directement avec la classe SQLConnection qui contient les trois méthodes classiques begin, commit et rollback.
La méthode ***begin*** est utilisée pour démarrer une nouvelle transaction avec le mode spécifié (voir ci-dessous).
La méthode ***commit*** est utilisée pour valider et terminer la transaction.
La méthode ***rollback*** est utilisée pour annuler la transaction.

Les différents modes de transaction sont les suivants :

- SQLTransactionLockType.DEFERRED : un verrou n'est pas acquis avant la première lecture ou écriture ;
- SQLTransactionLockType.EXCLUSIVE: un verrou est acquis dès que possible et aucune autre instance de la classe SQLConnection ne peut lire ou écrire dans la base de données ;
- SQLTransactionLockType.IMMEDIATE : un verrou est acquis dès que possible et les autres instances de la classe SQLConnection peuvent uniquement lire la base de données.

Les transactions ne sont pas limitées aux opérations effectuées sur une seule base, elles peuvent contenir des opérations effectuées sur différentes bases attachées à la connexion.

Exemple

```
try
{
  connection.begin(SQLTransactionLockType.EXCLUSIVE);

  // Opérations ...

  connection.commit();
}
catch(error:SQLError)
{
  connection.rollback();
  trace("Erreur : " + error.toString());
}
finally
{
  connection.close();
}
```

7. Pagination

Lorsqu'une base contient beaucoup de données, il est souvent nécessaire de ne récupérer qu'une partie des résultats, par exemple pour l'affichage dans une liste.

Il y a deux façons de procéder, la première est d'utiliser le mécanisme de la classe SQLStatement qui permet de limiter le nombre d'enregistrements retournés en passant ce nombre en paramètre de la méthode ***execute***. Les enregistrements suivants sont récupérés avec l'appel à la méthode ***next***. La propriété ***complete*** indique si l'ensemble des enregistrements a été retourné.

L'inconvénient de cette fonctionnalité est qu'on ne peut pas contrôler l'ordre dans lequel les enregistrements sont récupérés.

L'autre possibilité est de modifier directement la requête SQL pour filtrer les éléments retournés.
Les mots clés ***LIMIT*** et ***OFFSET*** permettent d'effectuer une pagination personnalisée.

L'exemple suivant permet de récupérer les cinq premiers utilisateurs :

Exemple

```
statement.text = "SELECT * FROM Utilisateur LIMIT @count OFFSET @start";
statement.parameters["@count"] = 5;
statement.parameters["@start"] = 0;
```

8. Introspection de la base

La classe SQLConnection permet de récupérer le schéma de la base (SQLSchemaResult) contenant la liste et la description des tables (SQLTableSchema), des colonnes (SQLColumnSchema), des vues (SQLViewSchema) et des triggers (SQLTriggerSchema).

L'exemple suivant montre comment afficher la liste des tables :

Exemple

```
connection.loadSchema();
var schema:SQLSchemaResult = connection.getSchemaResult();
for each (var table:SQLTableSchema in schema.tables)
{
  trace(table.name);
}
```

9. Types de données

Les différents types à utiliser pour définir les colonnes des tables sont les suivants :

- TEXT (ou STRING) ;
- NUMERIC ;
- INTEGER ;
- REAL (ou NUMBER) ;
- BOOLEAN ;
- DATE ;
- XML ;
- XMLLIST ;
- OBJECT ;
- NONE.

Il est possible de stocker des objets complexes avec le type OBJECT, il suffit d'ajouter à la classe la métadonnée [RemoteObject].
L'objet sera automatiquement sérialisé au format AMF3.

Le tableau ci-dessous indique la correspondance avec les types AS3 les plus communs :

AIR SQLite	Actionscript 3
TEXT	String
INTEGER	int / uint
REAL	Number
BOOLEAN	Boolean
DATE	Date
XML	XML
XMLLIST	XMLList
OBJECT	Object / Array / ByteArray / *

Le type NUMERIC accepte les types uint, int ou Number et retourne un type différent suivant la valeur.
Par exemple, si l'objet initial est de type Number et vaut 2, le type récupéré par la suite sera uint.

Vous pouvez consulter la documentation sur les types de données SQLite dans AIR pour plus d'informations : Lien41.

10. Conclusion

L'utilisation de SQLite dans une application AIR est très simple et rapide à mettre en place. Une utilisation poussée est aussi possible grâce aux nombreuses fonctionnalités présentées dans cet article.

Liens utiles :

- présentation AIR : Lien42 ;
- le forum SQLite : Lien43 ;
- le forum Flex : Lien44.

Consultez aussi le tutoriel et créez une application multiplateforme AIR avec SQLite : Lien40.

Retrouvez l'article de Julien Fray en ligne : Lien45

Visual C++ 2010 / C++ 0X : utilisation de nullptr

Dans mon billet précédent, je présentais mon article sur les nouveautés de Visual C++, avec notamment un point sur le support partiel de **C++0x** (Lien46).
Je vous propose de découvrir quelques-unes de ces nouvelles fonctionnalités en commençant par : **nullptr.**
Nullptr, littéralement : pointeur nul, devient un mot clef du langage à part entière, il est destiné à remplacer dans votre code NULL qui est une définition :

```
#define NULL 0
```

Nous pouvons maintenant écrire :

```
CData* pData= nullptr;
if ( pData == nullptr ){}

BOOL CMainFrame::CreateOutlookBar(…)
{
If(this==nullptr || m_hWnd== nullptr) return
FALSE;
}
```

Jusqu'à présent rien de bien essentiel par rapport à l'utilisation de **NULL**.
En fait ce mot clef va s'avérer utile dans le cas suivant :

```
void MyClass ::FunctionAdd(int *pInt){...
void MyClass ::FunctionAdd(int n){...
…
MyClass var;
var.FunctionAdd(nullptr); // Appellera
FunctionAdd(int *pInt);
var.FunctionAdd(NULL); // Appellera
FunctionAdd(int n); sans avertissement du
compilateur...
```

nullptr permet ici d'éviter la mauvaise résolution d'appel de la méthode suggérée par **NULL** qui est un entier.
Dernier point : Je n'ai pas fait l'essai, mais il serait intéressant de voir si **nullptr** peut être utilisé dans du code mixte **C++** et **C++CLI** qui utilise aussi ce mot clef…

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien47

Visual C++ 2010 / C++ 0X: utilisation de static_assert

Après avoir abordé nullptr (Lien47) dans mon dernier billet, passons à **static_assert**.
static_assert permet de vérifier une condition au moment de la compilation.
Sa syntaxe est la suivante :

```
static_assert( expression, message)
```

Exemple :

```
static_assert(0==1, "zero n'est pas egal a 1 !");
```

si la condition est vérifiée le compilateur ne dit rien, par contre en cas d'erreur comme dans l'exemple ci-dessus il génère une erreur **C2338** avec le message mentionné :
error C2338: zero n'est pas egal a 1 !
Le message ne supporte que les caractères de base et donc pas les accents.
Dans le cas d'utilisation de **static_assert** avec une classe **template**, la condition sera vérifiée lorsqu'une classe utilisant le **template** sera déclarée.
Référence MSDN : Lien48

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien49

Visual C++ 2010 / C++ 0X: Utilisation des références R-Value

Après avoir abordé static_assert (Lien49) dans mon dernier billet, passons à la référence **R-Value**.
Pour bien comprendre ce qui va suivre, il est impératif de connaître les méthodes de transmission d'un argument dans une méthode, en l'occurrence le passage par valeur et par référence.
Dans la littérature C++, **rValue** désigne un objet pouvant figurer à droite (right) d'une affectation, alors que **lValue** désigne un objet pouvant se trouver à gauche.
Concrètement une nouvelle forme de déclaration utilisant l'opérateur **&&** va nous permettre de transférer la responsabilité d'une ressource d'un objet temporaire à un autre, évitant ainsi une libération mémoire, une nouvelle allocation et la copie de la ressource concernée.
Voici une petite classe d'exemple qui illustre le sujet :

```
class XDate
{
public:
  XDate():m_szDate(nullptr){}
  ~XDate() {delete [] m_szDate;}

  XDate(const char *szDate):m_szDate(nullptr)
  {
    SetDate(szDate);
  }
  XDate(const XDate &date):m_szDate(nullptr)
  {
    SetDate(date.m_szDate);
  }
  void SetDate(const char *szDate)
  {
    if(m_szDate==nullptr) m_szDate= new char
[7];
    strcpy_s(m_szDate,7,szDate);
  }
  const XDate& operator=(const XDate& Src)
  {
    if(&Src==this) return *this;
```

```
    SetDate(Src.m_szDate);
    return *this;
  }

  const char *c_str(){return m_szDate;}

  void DebugTrace()
  {
    TRACE("\nDate:%04X
->(%s)",m_szDate,m_szDate==nullptr?"":m_szDate);
  }
public:
  char *m_szDate;
};

XDate GetCurrentDay()
{
  return XDate("030510");
}

void SetDate(const XDate& date)
{
  XDate d(date);

  d.DebugTrace();
}


int main()
{
  SetDate( GetCurrentDay());
}
```

XDate alloue en mémoire une chaîne de caractères pour tenir une date.
Dans l'appel réalisé dans le main on retrouve bien la séquence décrite précédemment où un objet temporaire fournit par **GetCurrentDay()** sert affecter un objet de la même classe dans la méthode **SetDate**.
Le nouvel opérateur **&&** va nous permettre de faire l'économie d'une libération, allocation et d'un transfert mémoire et je vais modifier pour l'exemple la fonction **SetDate** comme suit :

```
void SetDate(XDate&& date)
{
  XDate d;
  d.m_szDate=date.m_szDate;
  date.m_szDate=nullptr;

  d.DebugTrace();
}
```

Dans **SetDate** je récupère directement l'adresse contenue par le pointeur de l'objet temporaire et je positionne à **nullptr** celui de l'objet temporaire.
La méthode **SetDate** modifiée a été appelée parce que nous somme bien en présence d'un objet temporaire fournit par **GetCurrentDay()**.
Si je modifie comme suit la fonction **main()** :

```
int main()
{
  XDate d("030510");
  SetDate(d);
}
```

J'obtiens l'erreur :

```
error C2664: 'SetDate' : cannot convert parameter
1 from 'XDate' to 'XDate &&'
  You cannot bind an lvalue to an rvalue
reference
```

Note: IntelliSense m'avait déjà prévenu de l'erreur avant la compilation …
Si on veut avoir un code opérationnel dans les deux cas, il faudra laisser l'autre méthode **SetDate** active :

```
void SetDate(const XDate& date)
{
  XDate d(date);

  d.DebugTrace();
}
void SetDate(XDate&& date)
{
  XDate d;
  d.m_szDate=date.m_szDate;
  date.m_szDate=nullptr;
  d.DebugTrace();
}
```

Si on veut forcer la main au compilateur pour utiliser la méthode avec la déclaration **rValue** on pourra utiliser un **static_cast** :

```
int main()
{
  XDate d("030510");
  SetDate(static_cast<XDate &&>(d));
  d.DebugTrace();
}
```

Autre exemple :

```
int main()
{
  XDate d("030510");
  SetDate(d);
  SetDate(XDate("030510"));
}
```

Le premier SetDate reçoit une variable locale (lValue) et donc appellera la version const de SetDate.
Le deuxième appel est réalisé avec un objet temporaire car référencé nulle part dans le programme, c'est donc bien un rValue qui appellera la version XDate && de SetDate.
Bien, maintenant que nous avons compris comment tirer parti de cette fonctionnalité, il reste à implémenter ce mécanisme directement dans notre classe en ajoutant un constructeur de déplacement (Move constructor) et l'opérateur d'affectation (Move assignement) ainsi qu'un opérateur +.
La classe finale :

```
class XDate
{
public:
  XDate():m_szDate(nullptr){}
  ~XDate() {delete [] m_szDate;}

  XDate(const char *szDate):m_szDate(nullptr)
```

```
  {
    SetDate(szDate);
  }

  XDate(XDate &&date):m_szDate(nullptr)
  {
    Move(date);
    TRACE("\nMC:Date:%04X
->(%s)",m_szDate,m_szDate==nullptr?"":m_szDate);
  }

  XDate(const XDate &date):m_szDate(nullptr)
  {
    SetDate(date.m_szDate);
  }

  void SetDate(const char *szDate)
  {
    if(m_szDate==nullptr)  m_szDate= new char
[7];

    strcpy_s(m_szDate,7,szDate);
    DebugTrace();
  }
  XDate& operator=(XDate&& Src)
  {
    if(&Src==this) return *this;
    Move(Src);
    TRACE("\nMA:Date:%04X
->(%s)",m_szDate,m_szDate==nullptr?"":m_szDate);
    return *this;
  }

  friend XDate operator+(XDate&& left, const int
nDayRight);

  const XDate& operator=(const XDate& Src)
  {
    if(&Src==this) return *this;
    SetDate(Src.m_szDate);
    return *this;
  }

  const char *c_str(){return m_szDate;}

  void Move(XDate &date)
  {
    m_szDate=date.m_szDate;
    date.m_szDate=nullptr;
  }
  void DebugTrace()
  {
    TRACE("\nDate:%04X
->(%s)",m_szDate,m_szDate==nullptr?"":m_szDate);
  }
public:
  char *m_szDate;
};

XDate operator+(XDate && left, const int
nDayRight)
{
  XDate date;
  date.Move(left);
  TRACE("\nM+:Date:%04X
->(%s)",date.c_str(),date.c_str());
  // operation d'ajout du jour non
implementée...
  return date;
}
```

Et l'utilisation :

```
XDate d(GetCurrentDay());
d= GetCurrentDay()+1;
d.DebugTrace();
```

La première ligne utilise le constructeur de copie normal et pas le constructeur de déplacement comme on pourrait s'y attendre (ce qui ressemble bien à un bug (?)).
La seconde ligne va utiliser successivement l'opérateur +, le constructeur de déplacement et enfin l'opérateur d'affectation de déplacement.
Voila pour ce billet un peu long qui j'espère vous a éclairé sur l'utilisation de la référence **R-Value**.
Référence MSDN : Lien50

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien51

Visual C++ 2010 / C++ 0X: Utilisation du mot clef auto

Après avoir abordé la référence r-Value (Lien51) dans mon dernier billet, passons au mot-clef **auto**.
auto permet de déclarer une variable sans spécifier son type de donnée.
Un exemple très simple :

```
auto myvar= 1 ;
```

La ligne ci-dessus déclare un variable sans préciser son type.
Celui-ci est déterminé par le compilateur en examinant la partie droite de l'expression.
Le code est alors transformé en :

```
int myvar= 1 ;
```

Donc, s'il n'y a pas d'affectation le compilateur ne peut pas déterminer le type et retournera une erreur.
Le point important qu'il faut retenir c'est que le type de donnée est déterminé à la compilation et non à l'exécution du code.
Dans mon article sur Visual studio 2010 (Lien46) j'avais évoqué les améliorations d'IntelliSense, dans le contexte présent celui-ci nous fournit directement le type de donnée qui sera associée à la variable, il suffit de survoler la souris sur le nom de la variable.
On pourra donc déclarer toutes sortes de variables : des pointeurs, des tableaux de pointeurs des constantes, tout cela est possible à partir du moment ou le compilateur peut déterminer le type associé dans la déclaration.

```
auto dArray= new double[5]; //tableau de double *
auto pMyVar=&myvar ; // un pointeur sur myvar.
const auto nMax=100 ; // une constante.
```

Etc.
Ce qui ne peut être géré.
L'utilisation d'**auto** pour le retour d'une fonction :

```
auto getval();
```

L'utilisation d'**auto** pour l'argument d'une fonction :

```
int getval(auto n) ;
```

La déclaration d'une variable avec **auto** dans la déclaration d'une classe ou d'une structure.
Les déclarations multiples sur une ligne qui donnent lieu à des types différents :

```
auto n=1,d=1.0 ,l=1L ;
```

Après ce tour d'horizon des fonctionnalités basiques de ce mot clef, examinons maintenant ce pour quoi il va être vraiment utile.
Vous avez certainement tous utilisé les conteneurs de la STL (j'espère !) , en voici un exemple tiré de mes codes :

```
// dans le .h de ma classe :
std::vector<CItem> m_Items;
// dans mon code:

std::vector<CItem>::iterator it;

for(it=m_Items.begin();it!=m_Items.end();++it)
  if(it->m_nID==nID)
    return (it->m_dwFlags & itemChecked)!=0;
```

auto va me permettre de me libérer de la syntaxe lourde de l'itérateur (qui est ici très simple) :

```
for(auto it=m_Items.begin();it!=m_Items.end();+
+it)
  if(it->m_nID==nID)
    return (it->m_dwFlags & itemChecked)!=0;
```

C'est plus agréable à écrire non ?
Si j'ai besoin d'un **const_iterator**, la classe vector y pourvoit :

```
for(auto it=m_Items.cbegin();it!=m_Items.cend();+
+it)
  if(it->m_nID==nID)
    return (it->m_dwFlags & itemChecked)!=0;
```

le couple **begin() end()** laisse la place à **cbegin() cend()**.
Mon exemple était volontairement simple, mais je pense que tout le monde appréciera la simplicité retrouvée pour déclarer un itérateur sur un objet conteneur.
Référence MSDN : Lien52

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien53

Visual C++ 2010 / C++ 0X : utilisation de delctype

decltype permet de déduire le type de donnée d'une expression :

```
int n ;
decltype(n) var ;
```

Déclare la variable **var** comme étant un entier, le compilateur connaît le type de donnée **n** et remplacera l'expression par :

```
int var ;
```

Par rapport à auto (Lien53) que nous avons vu dans le billet précédent, on peut se poser la question de l'intérêt de **decltype**, en fait celui-ci vient en complément de **auto**.
Considérons le cas suivant :

```
auto d=func() ;
```

Je viens de déclarer une variable **d** en accord avec le retour de **func()**.
Maintenant si je veux déclarer une autre variable du même type que **d**, je ne veux pas ou ne peut pas forcément appeler la même fonction pour déclarer ma variable.
Dans ce cas l'utilisation de **delctype** résout le problème :

```
auto d=func() ;
delctype(d) dd ;
```

Je peux aussi écrire :

```
decltype(func()) dd ;
```

Car à la différence de **auto, decltype n'exécute pas l'expression,** il analyse le type de donnée de l'expression.
Comme précédemment avec **auto**, **IntelliSense** est capable de nous donner l'information sur le type de donnée de la variable ainsi déclarée.
Référence MSDN : Lien54

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien55

Visual C++ 2010 / C++ 0X: utilisation du Trailing return type

Après avoir abordé decltype (Lien55) dans mon dernier billet, passons au **"Trailing return type"** ou en français la mise en place du type de retour (je n'ai pas trouvé mieux comme traduction).

Cette syntaxe, utilisée dans les expressions lambda, permet de spécifier le type de retour d'une fonction.
Considérons les deux exemples qui suivent
L'expression la plus simple :

```
auto GetLong() -> long
{
  return 1L;
}
```

J'ai spécifié que le type de retour était un long grâce à la syntaxe **-> type**
Bien entendu, cet exemple comme celui qui va suivre, est franchement improbable dans notre codage de tous les jours :

```
auto GetFloat() -> decltype(1.0)
{
  return 1.0;
}
```

Cette fois-ci le retour est spécifié par l'évaluation de l'expression contenue dans **decltype**.
Alors franchement à quoi cela peut-il servir ?
Cette syntaxe peut avoir son utilité avec l'utilisation des modèles (templates).
Par exemple avec cette fonction template que j'écris initialement comme suit :

```
template <typename Arg1, typename Arg2>
Arg2 AddItem(Arg1 a1, Arg2 a2)
{
  return a1 + a2;
}
```

Cette fonction se propose d'additionner deux types de données et pour spécifier le retour de ma fonction, j'ai spécifié de manière arbitraire le type Arg2, ce qui réduit fortement les possibilités d'utilisation de ma fonction.
Si je reprends mon exemple sur les rvalues (Lien51) avec la classe **XDate** et que je veuille écrire ce genre de chose :

```
XDate d(GetCurrentDay());

auto result=AddItem(d, 1);
```

cet exemple ne passera pas à la compilation.

```
template <typename Arg1, typename Arg2>
auto AddItem(Arg1 a1, Arg2 a2) ->
decltype(a1+a2)
{
  return a1 + a2;
}
```

Celui-ci par contre, fonctionne très bien, auto me permet de rester neutre sur le retour qui sera évalué par decltype.
Pour être complet dans mon exemple j'ai dû rajouter l'opérateur + dans ma classe **XDate** pour que l'exemple compile.

```
class XDate
{
//…..
  friend XDate operator+(XDate&& left, const int
nDayRight);
  friend XDate operator+(const XDate& left, const
int nDayRight);
//…
};
XDate operator+(const XDate & left, const int
nDayRight)
{
  XDate date;
// …
  TRACE("\nM+:Date:%04X
->(%s)",date.c_str(),date.c_str());
  // operation de concatenation du jour non
implementée...
  return date;
}
```

Dans cet exemple on voit la puissance du mécanisme puisque l'expression spécifiée dans **decltype** a permis au compilateur de déterminer le retour puisque l'opérateur + **de XDate** était disponible.
L'exemple suivant fonctionne aussi sans avertissement de conversion :

```
auto v=AddItem(10.,1);
```

J'additionne un **double** et un **entier**.
Pour finir j'ai constaté que dans cet exemple **IntelliSense** ne savait pas trop comment s'en sortir avec cette utilisation.

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien56

Visual C++ 2010 / C++ 0X: utilisation des expressions lambda

Après avoir abordé le Trailing return type (Lien56) dans mon dernier billet, passons aux "Expressions lambda".
Considérons la classe ci-dessous :

```
class Adherent
{
public:
  Adherent(LPCTSTR szNom,LPCTSTR
szPrenom,const int nAge)
  {
   if(szNom==nullptr ||
szPrenom==nullptr || nAge<=0)
        AfxThrowInvalidArgException();
   m_strNom=szNom;
   m_strPrenom=szPrenom;
   m_nAge=nAge;
  }
public:
  CString m_strNom,m_strPrenom;
  int m_nAge;
};
```

Cette classe très simple permet d'enregistrer les informations d'un adhérent.
Supposons maintenant que je maintienne en mémoire un tableau de mes adhérents et que je veuille le trier selon l'âge des adhérents.
Je vais « naturellement » utiliser la classe **vector** et un foncteur pour trier mon **vector**.
Le code ressemblera donc à ceci :

```
struct SortAdherentAge
{
  bool operator()(const Adherent &ad1,const
Adherent &ad2) const
  {
    return ad1.m_nAge < ad2.m_nAge;
  }
};
```

L'utilisation plus loin dans le code donnera ceci :

```
  std::vector <Adherent> vAdherent;

  vAdherent.push_back(Adherent(_T("Marcel"),_T("dupont"),50));
  vAdherent.push_back(Adherent(_T("alain"),_T("machin"),60));
  vAdherent.push_back(Adherent(_T("patrick"),_T("meyer"),30));

  std::sort(vAdherent.begin(),vAdherent.end(),SortAdherentAge());
```

Les foncteurs sont un moyen très puissant pour personnaliser les algorithmes standards de la **STL**, malheureusement leur définition est peu pratique, car elle nécessite d'écrire une classe entière comme **SortAdherentAge** dans mon exemple.
Le second défaut est que cette classe n'est pas définie dans le code source, à l'emplacement où on va l'utiliser, ce qui rend leur utilisation plus difficile.
C'est là qu'interviennent les expressions lambda qui vont gommer tous ces défauts.

Reprenons maintenant le même exemple :

```
std::vector <Adherent> vAdherent;

  vAdherent.push_back(Adherent(_T("Marcel"),_T("dupont"),50));
  vAdherent.push_back(Adherent(_T("alain"),_T("machin"),60));
  vAdherent.push_back(Adherent(_T("patrick"),_T("meyer"),30));

  std::sort(vAdherent.begin(),vAdherent.end(),
  [](const Adherent &ad1,const Adherent &ad2)->bool{ return ad1.m_nAge < ad2.m_nAge;});
```

L'expression lambda débute avec l'écriture de **"[]"**, la déclaration des paramètres **ad1** et **ad2** fournit au compilateur les paramètres de la pseudo-fonction anonyme.
Le type de retour de la fonction est précisé par **« ->bool »**, le corps de la fonction est exprimé entre les deux accolades.
Avec les expressions lambda parcourir le tableau et réaliser un traitement **« inline »** devient beaucoup plus facile :

```
  for_each(vAdherent.begin(),vAdherent.end(),
    [](const Adherent &ad)
    {
     TRACE("\n %d",ad.m_nAge);
    }
     );
```

Consultez MSDN (Lien57) pour plus de renseignements sur les syntaxes possibles.

Retrouvez ce billet sur le blog de Patrick Ottavi : Lien58

[Qt DevDays 2010] Participation de Développez.com et présentation des conférences

Bienvenue aux *Qt Developer Days 2010* !

Les *Qt Developer Days* sont le seul événement où vous pouvez apprendre, directement de la bouche des développeurs du framework, la meilleure manière de l'utiliser avec son EDI de référence, Qt Creator, pour atteindre les millions d'utilisateurs de Symbian, de MeeGo, de Linux et de tous les systèmes d'exploitation, embarqués ou non, avec des applications innovantes.

Qt est l'un des frameworks qui se développent le plus vite dans le monde, avec des centaines de milliers de développeurs l'utilisant activement. D'importantes compagnies comme Alcatel-Lucent, AMD, Google, Oracle, Samsung, Siemens, Lucas Film Entertainment Company Ltd. et l'Agence spatiale européenne utilisent Qt pour fournir des applications majeures sur une multitude de plateformes.

Vous pourrez y rencontrer et discuter avec les développeurs de Qt, améliorer vos capacités de développement avec Qt avec des sessions qui expliquent les entrailles du framework, des laboratoires, découvrir comment utiliser Qt pour cibler Symbian, MeeGo et toutes les plateformes majeures, identifier de nouvelles possibilités pour atteindre des millions d'utilisateurs avec votre application et vous engager avec toute la communauté Qt.

Les dates :
Munich, Germany : du 11 au 13 octobre
Économisez 200 € si vous vous y inscrivez avant le 15 septembre !
San Francisco, California : du 1 au 3 novembre
Économisez 200 $ si vous vous y inscrivez avant le 29 septembre !

Inscrivez-vous aux Qt Developer Days 2010 : Lien59

Allez-vous participer à cet événement ? Quelles conférences seront sur votre liste ?

Participation de Développez.com et présentation des conférences

Bonjour à tous,
cette année encore, l'équipe Qt de Developpez.com répondra présent aux Qt Dev Days 2010 à Munich qui se dérouleront cette année du 11 au 13 octobre.
À cette occasion, nous avons réalisé une présentation de l'ensemble des conférences permettant de savoir ce qui va être présenté et, si jamais vous avez la chance d'y participer également, de faire votre choix parmi celles auxquelles vous désirez vous rendre.

Présentation des Qt DevDays 2010 : Lien60

Ce sera également pour moi, en tant que reporter, l'occasion de rencontrer certains trolls de chez Nokia, Qt Developpement framework et de leur poser des questions sur le présent et l'avenir de Qt. Même si déjà certaines questions trottent dans ma tête, à vous de me dire également ce que vous aimeriez savoir ?

Si vous êtes un habitué de la rubrique Qt de Developpez.com et que vous allez participer aux Qt DevDays 2010 à Munich, cela pourrait être l'occasion de me rencontrer et de discuter (en français) de Qt et de la rubrique.

Serez-vous présent à l'édition des Qt DevDays 2010 à Munich ? Quelles conférences vous semblent les plus intéressantes ? Quels questions poseriez-vous aux développeurs du framework Qt si vous pouviez les interviewer ?

Les Qt DevDays 2010 sur le site de Nokia : Lien59

Commentez cette news de Thibaut Cuvelier en ligne : Lien61

Qt 4.7 disponible en téléchargement

Qt 4.7 est sorti et est maintenant disponible en téléchargement sur http://qt.nokia.com (Lien62).

Cette nouvelle version de Qt donne aux développeurs tout ce dont ils ont besoin pour créer des applications riches avec la possibilité d'interfaces graphiques tactiles pour toutes les plateformes supportées par Qt. Neuf mois après la sortie de Qt 4.6.0, le besoin de nouvelles fonctionnalités, d'innovation se faisait déjà sentir.

Qt Declarative

Qt 4.7 fournit comme nouvelle fonctionnalité principale le QML, le nouveau langage déclaratif facile à apprendre, accompagné du module Qt Declarative, le support technique de QML. Ce langage supporte aussi le JavaScript pour les habitués du scriptage d'interfaces.

Cette idée a germé dans l'esprit des développeurs il y a deux ans, peu après la sortie de Qt 4.4. Deux années auront été nécessaires pour fournir un produit pleinement mature, pleinement fonctionnel. Rappelons qu'un de ses objectifs est de rapprocher les développeurs des designers d'interfaces, tous les détails ont déjà été écrits (Lien63).

La dernière brique de l'ensemble, le designer pour Qt Creator, sera disponible dans Qt Creator 2.1, à paraître en 2010

Les performances

Pour QtWebKit, le compositing accéléré matériellement a augmenté de 61 % le rendu des animations. Le défilement a aussi été complètement revu : sur des sites très complexes comme Facebook, il n'a été amélioré que de... 67 % ! Sur des sites plus simples, le gain peut atteindre 350 % !

Pour le rendu de texte, la nouvelle classe QStaticText offre un rendu deux fois plus rapide que ce qui était possible auparavant. Le moteur de QPainter rend possible un rendu plus efficace des systèmes de particules avec OpenGL.

Le mot du vice-président

Sebastian Nyström, Vice President, Application and Service Frameworks, Nokia

```
Qt 4.7 is an important step forward that keeps Qt
at the forefront of UI and application
development frameworks.

Developers looking to create rich, fluid UIs and
apps will be amazed at how easy it is to use the
new features in Qt 4.7.

Continuing to enhance Qt's performance and
stability is vital, and we're proud to have taken
Qt even further in these two areas.
```

Améliorations diverses

Cette nouvelle version comprend également de meilleures performances notamment au sein de QtWebKit (jusque 350 % plus rapide que la version disponible dans Qt 4.6.0 !), une mise à jour de l'add-in Visual Studio et il s'agit également de la première version majeure de Qt à avoir de nouveaux critères de performance et de stabilité.

Il faut aussi remarquer que l'essentiel des corrections a pu être apporté grâce aux retours de la communauté depuis la version RC, disponible depuis fin août. Le nouveau modèle de contribution de Qt rend en effet beaucoup plus facile la participation active au code (vous pouvez envoyer des patchs *via* Gitorious, par exemple).

Last but not least, le KDE Platform va très bientôt utiliser cette version 4.7 pour bénéficier de toutes les améliorations désormais finalisées !

Le lien du jour

Téléchargez-le aujourd'hui ! : Lien64

Le Qt SDK a été mis à jour avec cette dernière version, la version 2010.5 est disponible dès aujourd'hui à l'adresse ci-dessus.

Pour en savoir plus sur Qt 4.7, rendez-vous sur What's New? (Lien65) ou participez au Qt Developer Days 2010 (Lien59)

Source : communiqué de presse

Voir aussi

Les développeurs viennent de Mars, les designers de Vénus (Lien66), un article de Hietala Nigel paru dans la Qt Quarterly Issue 33 (Lien67)

Commentez cette news de Jonathan Courtois en ligne : Lien68

Traductions des Qt Labs, lieu d'expérimentation par excellence des développeurs de Qt

La rubrique Qt continue de s'agrandir !

En plus des rubriques habituelles proposées (des articles (Lien69), une FAQ (Lien70), les principaux outils (Lien71), des critiques de livres (Lien72), un blog (Lien73), des binaires compilés par l'équipe (Lien74)) et des nombreuses traductions (la documentation de Qt (Lien75), Qt Quarterly (Lien76)), l'équipe Qt de Developpez.com est fière de vous présenter notre nouveau domaine Qt.

Les traductions de Qt Labs : Lien77

Ce nouveau domaine regroupe les traductions des articles de Qt Labs, le blog des développeurs de Qt. On y retrouve des articles sur les techniques avancées utilisant ou utilisées dans Qt. Il permet de partager les idées et des composants et d'obtenir des retours d'informations.

Actuellement, vous y trouverez plusieurs articles, dont quelques inédits, traduits à l'occasion de l'ouverture de ce domaine. De nombreux autres articles sont prévus dans les semaines à venir !

Bonne lecture !

Commentez cette news de Guillaume Belz en ligne : Lien78

Intégration d'OpenGL dans une interface Qt

À travers ce tutoriel vous allez apprendre à intégrer simplement un widget OpenGL dans une application Qt.

1. Introduction

Qt propose un module OpenGL qui permet d'afficher des rendus OpenGL dans une fenêtre Qt. Toutes les versions de Qt disposent de ce module. Qt wrappe OpenGL c'est-à-dire que l'utilisation du module OpenGL ne varie pas selon les versions d'OpenGL. Vous pouvez ainsi mettre à jour vos en-têtes OpenGL sans souci de compatibilité avec Qt.

1.1. De quelle version d'OpenGL Qt dispose-t-il ?

La version d'OpenGL varie selon les compilateurs.

1.2. Qu'y a-t-il à apprendre dans ce tutoriel ?

Eh bien, il est évident que nous n'allons pas apprendre à utiliser OpenGL de façon poussée puisque le tutoriel porte essentiellement sur l'interaction entre Qt et OpenGL. OpenGL ne disposant pas d'interface graphique (fenêtres, boîtes de dialogue) en soi, cette bibliothèque permet seulement d'afficher des formes soumises à des transformations dans l'espace. Qt nous permettra donc de créer la fenêtre qui contiendra la zone d'affichage OpenGL. Je vous apprendrai à fermer la fenêtre en appuyant sur une touche, afficher la fenêtre en plein écran, charger des textures, etc.

1.3. En quoi utiliser OpenGL avec Qt peut-il être intéressant ?

Eh bien, Qt à l'avantage de conserver le côté multiplateforme d'OpenGL mais aussi simplifie certaines parties assez fastidieuses telles que le chargement d'une image pour l'appliquer en tant que texture ou même l'intégration des shaders. L'interface graphique de Qt est très complète comparée à d'autres bibliothèques comme SDL et GLUT très utilisées avec OpenGL en général, ainsi elle permet la réalisation de logiciels tels qu'un éditeur de cartes pour votre jeu, modélisateur 3D, etc. Tous ces exemples sont autant de points forts qui justifient l'utilisation d'OpenGL avec Qt. Maintenant que vous savez de quoi le tutoriel traite, nous allons pouvoir nous plonger dans un nouveau projet Qt. L'IDE que j'utilise est QtCreator 2.0, si vous en utilisez un autre cela reste très similaire.

2. Prérequis

Comme il a déjà été dit dans l'introduction, ce n'est pas un tutoriel sur OpenGL mais plutôt un tutoriel sur l'interaction entre Qt et OpenGL. Il est donc nécessaire que vous ayez des connaissances en OpenGL. Ce n'est pas non plus un tutoriel sur l'apprentissage de Qt, vous devez avoir déjà des connaissances sur Qt, comme créer les fenêtres, les signaux et les slots. Tout d'abord, créer un nouveau projet vide, vous y ajoutez un fichier source main.cpp comme pour tous vos autres projets Qt habituels avec le code minimal :

```
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    return app.exec();
}
```

Nous allons maintenant configurer le fichier .pro de Qt pour permettre au compilateur d'ajouter le module OpenGL en ajoutant cette ligne :

```
QT += opengl
```

Voilà, notre projet est configuré pour pouvoir traiter du code OpenGL avec Qt. Il est inutile de lancer le programme puisqu'il n'exécute pas de fenêtre.

3. Première approche de QGLWidget

La classe QGLWidget (Lien79) est le widget de Qt permettant d'afficher des rendus OpenGL. L'utilisation reste simple et il suffira de créer une classe héritant de celle-ci. Cette classe est constituée notamment de trois fonctions très importantes que nous allons utiliser dans ce tutoriel.

- paintGL() (Lien80) : méthode appelée à chaque fois que le widget doit être mis à jour. Elle redessine la scène OpenGL.
- resizeGL(int width, int height) (Lien81) : méthode appelée à chaque fois que le widget est redimensionné. Cette méthode met à jour la scène OpenGL quand la fenêtre est redimensionnée.
- InitializeGL() (Lien82) : méthode permettant d'initialiser les différents paramètres d'OpenGL. Cette méthode est appelée une fois au lancement de l'application avant même paintGL() et resizeGL(int width, int height).

3.1. myGLWidget.h

Maintenant que nous avons vu et compris ces trois fonctions, nous allons pouvoir nous concentrer sur le code.

```
#ifndef MYGLWIDGET_H
#define MYGLWIDGET_H

#include <QtOpenGL>
#include <QGLWidget>

class myGLWidget : public QGLWidget
{
    Q_OBJECT
public:
    explicit myGLWidget(int framesPerSecond = 0,
```

```
QWidget *parent = 0, char *name = 0);
    virtual void initializeGL() = 0;
    virtual void resizeGL(int width, int height)
= 0;
    virtual void paintGL() = 0;
    virtual void keyPressEvent( QKeyEvent
*keyEvent );

public slots:
    virtual void timeOutSlot();

private:
    QTimer *t_Timer;

};


#endif // MYGLWIDGET_H
```

Nous allons nous concentrer sur les parties importantes du code et nous allons les détailler afin de bien les comprendre.

```
#include <QtOpenGL>
#include <QGLWidget>
```

Ces inclusions sont nécessaires pour pouvoir utiliser le module OpenGL de Qt.

```
class myGLWidget : public QGLWidget
```

Notre classe va hériter de QGLWidget (Lien79) car elle servira au rendu OpenGL dans une fenêtre Qt.

```
explicit myGLWidget(int framesPerSecond = 0,
QWidget *parent = 0, char *name = 0);
```

Quelques explications s'imposent, tout d'abord framesPersecond représente le nombre d'images par seconde que l'utilisateur souhaite afficher, cela permettra de définir le temps, entre chaque image, rendu par OpenGL pour éviter de surcharger votre processeur et votre carte graphique inutilement. Ensuite parent est un pointeur sur le widget qui contiendra notre objet myGLWidget. Le dernier paramètre name concerne tout simplement le titre de la fenêtre affiché en haut de celle-ci.

```
virtual void initializeGL() = 0;
virtual void resizeGL(int width, int height) = 0;
virtual void paintGL() = 0;
```

Remarquons que ces fonctions sont virtuelles pures. Viendra plus tard une classe dérivant de myQGLWidget pour afficher notre scène OpenGL, mais nous y reviendrons dans le chapitre suivant.

```
virtual void keyPressEvent( QKeyEvent
*keyEvent );
```

Voilà une fonction importante. Son rôle ? Recevoir les informations sur les touches du clavier pressées par l'utilisateur. Il y a encore beaucoup plus à dire sur elle, nous y viendrons avec son implémentation dans le fichier myGLWidget.cpp.

```
public slots:
    virtual void timeOutSlot();

private:
    QTimer *t_Timer;

};
```

Le timer est nécessaire à notre application pour éviter de surcharger notre processeur et notre carte graphique. L'utilisation d'un Qtimer (Lien83) facilite la tâche. Notre processeur travaille tant qu'on lui donne des informations à calculer, donc sans timer, notre scène OpenGL pourra avoir un nombre d'images par seconde très élevé inutilement. Notre œil ne perçoit pas la différence entre 60 images par seconde (ips) et 999 images par seconde. Ainsi il est inutile de demander à notre processeur et à notre carte graphique d'afficher 999 ips voir plus (ce n'est qu'un exemple) alors que 60 ips suffisent largement. Nous choisirons 60 ips puisqu'en dessous on perçoit une légère latence qui fatigue les yeux à la longue mais rien ne nous empêche de définir une valeur différente. Pour le moment ces informations ne sont données qu'à titre indicatif puisque nous allons définir cette valeur dans le chapitre suivant.

3.2. myGLWidget.cpp

Le code complet est le suivant :

```
#include "myGLWidget.h"

myGLWidget::myGLWidget(int framesPerSecond,
QWidget *parent, char *name)
    : QGLWidget(parent)
{
    setWindowTitle(QString::fromUtf8(name));
    if(framesPerSecond == 0)
        t_Timer = NULL;
    else
    {
        int seconde = 1000; // 1 seconde = 1000
ms
        int timerInterval = seconde /
framesPerSecond;
        t_Timer = new QTimer(this);
        connect(t_Timer, SIGNAL(timeout()), this,
SLOT(timeOutSlot()));
        t_Timer->start( timerInterval );
    }
}

void myGLWidget::keyPressEvent(QKeyEvent
*keyEvent)
{
    switch(keyEvent->key())
    {
        case Qt::Key_Escape:
            close();
            break;
    }
}

void myGLWidget::timeOutSlot()
{
}
```

Comme dans la partie précédente, détaillons le code pour mieux le comprendre.

```
myGLWidget::myGLWidget(int framesPerSecond,
QWidget *parent, char *name)
    : QGLWidget(parent)
{
    setWindowTitle(QString::fromUtf8(name));
```

La fonction setWindowTitle() (Lien84) permet de définir le nom de la fenêtre qui sera le paramètre name.

```
  if(framesPerSecond == 0)
    t_Timer = NULL;
  else
  {
    int seconde = 1000; // 1 seconde = 1000 ms
    int timerInterval = seconde /
framesPerSecond;
    t_Timer = new QTimer(this);
    connect(t_Timer, SIGNAL(timeout()), this,
SLOT(timeOutSlot()));
    t_Timer->start( timerInterval );
  }
}
```

À cet endroit, un test est effectué sur la valeur de framesPerSecond : s'il vaut 0, ce qui est la valeur par défaut, alors notre QTimer t_Timer aura null pour valeur. Sinon, nous calculons l'intervalle entre deux images puis nous enregistrons cette valeur dans timerInterval, ensuite nous créons notre timer puis connectons le signal timeout() (Lien85) au slot personnalisé timeOutSlot(). Pour finir nous démarrons le timer avec la valeur de timerInterval. Le timer se met ainsi en marche, contrôlant le temps entre deux images rendu par OpenGL.

```
void myGLWidget::keyPressEvent(QKeyEvent
*keyEvent)
{
    switch(keyEvent->key())
    {
        case Qt::Key_Escape:
            close();
            break;
    }
}
```

La fonction keyPressEvent() (Lien86) va nous permettre comme dit précédemment de détecter l'appui sur une touche du clavier. Ainsi nous allons pouvoir traiter les événements relatifs au clavier.

Cette fonction reçoit en paramètre un QkeyEvent (Lien87) qui correspond à l'événement du clavier. Nous effectuons un switch qui nous permettra de traiter les événements selon leur nature, ici nous nous contenterons de fermer la fenêtre lors d'un appui sur la touche ‹ Echap ›.

```
void myGLWidget::timeOutSlot()
{
}
```

Ici, notre fonction timeOutSlot() qui est appelée par le timer, est vide pour le moment. On y ajoutera du code dans le chapitre suivant.

Jusqu'ici, rien de fonctionnel n'a été mis en place. Cependant, toute cette partie est requise, elle pose les bases sur lesquelles tout l'affichage s'appuie. Après cette longue partie théorique, passons à du plus ludique avec l'affichage de votre premier polygone avec OpenGL et Qt !

Notre classe myGLWidget va être elle-même dérivée dans la partie suivante. Pour quelle raison ? Eh bien, cette classe sera la classe mère pour tous les autres chapitres. Ainsi on aura par défaut (dans cette classe) des fonctionnalités importantes qu'il n'est pas nécessaire de redéfinir à chaque nouvelle partie du tutoriel telles que la possibilité de changer de mode d'affichage (plein écran/fenêtré), ou de fermer l'application en appuyant sur la touche ‹ Echap ›, etc.

4. Dessiner des objets OpenGL dans ma fenêtre

Ce chapitre va enfin nous amener vers la partie intéressante du tutoriel avec du résultat. Ajoutons deux fichiers à notre projet : myWindow.h et myWindow.cpp. Ces fichiers comporteront la classe myWindow dérivée de myGLWidget qui aura pour but d'afficher le rendu. Nous redéfinirons les fonctions virtuelles pures initializeGL() (Lien82), resizeGL() (Lien81) et paintGL() (Lien80). Dans ce chapitre, vous constaterez que l'utilisation d'OpenGL avec Qt reste identique pour ce qui est de la syntaxe, ainsi tout ce que vous avez appris sur le code strictement OpenGL vous sera indispensable pour comprendre certaines lignes non expliquées dans cette partie.

4.1. myWindow.h

```
#ifndef MYWINDOW_H
#define MYWINDOW_H

#include "myGLWidget.h"

class myWindow : public myGLWidget
{
    Q_OBJECT
public:
    explicit myWindow(QWidget *parent = 0);
    void initializeGL();
    void resizeGL(int width, int height);
    void paintGL();
};

#endif // MYWINDOW_H
```

4.2. myWindow.cpp

```
#include "myWindow.h"

myWindow::myWindow(QWidget *parent)
    : myGLWidget(60, parent, "Premier Polygone
avec OpenGL et Qt")
{
}

void myWindow::initializeGL()
{
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT,
```

```
GL_NICEST);
}

void myWindow::resizeGL(int width, int height)
{
    if(height == 0)
        height = 1;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (GLfloat)width/
(GLfloat)height, 0.1f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void myWindow::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f, 0.0f, -6.0f);

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
    glEnd();

    glTranslatef(3.0f, 0.0f, -6.0f);

    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3d(1.0f, 1.0f, 0.0f);
    glEnd();
}
```

Le code n'est pas vraiment compliqué ; certains points sont cependant assez importants et seront donc détaillés.

```
myWindow::myWindow(QWidget *parent)
    : myGLWidget(60, parent, "Premier Polygone
avec OpenGL et Qt")
{
}
```

Nous avons choisi de « brider » le rendu OpenGL à 60 ips pour éviter de surcharger le processeur et la carte graphique pour rien. Ainsi on attribue la valeur 60. Cette valeur est le nombre d'images par seconde. Le reste des arguments se passe d'explication.

```
void myWindow::initializeGL()
{
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT,
GL_NICEST);
}
```

Comme vous pouvez le constater, la fonction initializeGL() est tout à fait similaire aux fonctions d'initialisation OpenGL avec les autres bibliothèques telles que GLUT ou SDL. Rien de particulier donc à signaler.

```
void myWindow::resizeGL(int width, int height)
{
    if(height == 0)
        height = 1;
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (GLfloat)width/
(GLfloat)height, 0.1f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

Comme dans la plupart des programmes avec OpenGL, il faut gérer le redimensionnement de telle sorte que ça ne déforme pas la scène lors d'un changement de résolution de l'écran ou d'un redimensionnement de la fenêtre. Cette fonction a ce rôle et vous avez déjà utilisé une façon similaire pour parer à toute déformation non souhaitée de la scène OpenGL.

```
void myWindow::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f, 0.0f, -6.0f);

    glBegin(GL_TRIANGLES);
        glVertex3f(0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
    glEnd();

    glTranslatef(3.0f, 0.0f, -6.0f);

    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f, -1.0f, 0.0f);
        glVertex3f(1.0f, -1.0f, 0.0f);
        glVertex3d(1.0f, 1.0f, 0.0f);
    glEnd();
}
```

La fonction paintGL() permet d'afficher la scène OpenGL. Ici notre scène est constituée d'un triangle et d'un carré.

4.3. Modification du main.cpp

Pour terminer, afin que le programme soit fonctionnel, il nous suffit de retourner dans notre main.cpp et d'y ajouter quelques lignes de code.

```
#include <QApplication>
#include "myWindow.h"

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);
  myWindow myWin; //Ajout de notre classe
myWindow
  myWin.show();   //Exécution de notre fenêtre de
rendu OpenGL
  return app.exec();
}
```

Retrouvez la suite de l'article de Florentin Halgand en ligne : Lien88

Intégration de Qt 4 Open Source avec Apple Xcode 3.0 et 3.1

Qt est un toolkit fabuleux disposant de kits d'intégration pour divers IDE dans sa version commerciale mais quasiment aucun pour sa version Open Source. Ce tutoriel cherche à vous faciliter l'intégration de Qt avec Apple Xcode pour Mac OS X.
Ce tutoriel requiert l'installation de Qt 4 pour Mac OS X bien que cette étape sorte du cadre de cet exposé.

1. Les fichiers projets Qt

Afin de conserver une portabilité maximale au travers de systèmes aussi hétéroclites que Windows, Mac OS X ou Linux, Qt propose un ensemble d'outils de gestion des projets. La base de cet ensemble est représentée par un fichier se terminant par l'extension .pro (pour projet) et l'utilitaire qmake. Le principe est simple : la configuration du projet s'effectue au sein du fichier .pro et l'utilitaire qmake se charge de générer les scripts de compilation adéquats en fonction de la plateforme (notamment la création des fameux Makefile s).

Ce principe de fonctionnement est très spécifique à Qt et des environnements de développement intégrés (EDI en français, IDE en anglais) comme Xcode ne sont pas conçus pour le supporter dès leur mise en route. Il existe toutefois des solutions pour contourner une partie des écueils que l'on peut rencontrer dans ce cas précis en acceptant de faire une croix sur certaines facilités.

2. Projet Xcode ou projet Makefile ?

Deux approches sont à considérer en débutant un développement Qt sous Xcode : faut-il choisir de travailler uniquement à partir des projets natifs Xcode (fichiers .xcodeproj) ou d'envisager un projet à base d'un fichier .pro et de Makefiles ?

Autant vous le dire immédiatement, la première solution n'offre que peu d'avantages, puisque créer un fichier de projet Xcode pour gérer la totalité de votre développement ne permet pas de prendre en charge l'actualisation du contenu de votre fichier .pro lors de l'ajout de classes ou de boîtes de dialogues. Les fichiers .pro constituant la base d'un projet Qt, il vous serait alors impossible de transposer de façon immédiate vos fichiers sources depuis votre Mac vers n'importe quel autre système ou, plus ennuyeux, environnement de développement. C'est pourquoi nous choisirons la seconde approche.

3. Configurer un projet Makefile

Ce que nous appelons un « projet Makefile » est tout simplement une méthode détournée qui permet de configurer Xcode (ou n'importe quel autre IDE qui l'autorise) pour qu'il utilise un système de configuration à base de fichiers Makefile. L'intérêt, dans le cas de Qt, est que les fichiers .pro et qmake fonctionnent à merveille avec cette solution, de façon portable.

Nous envisagerons les étapes de façon différente pour Xcode 3.0 ou Xcode 3.1 (en version bêta à l'heure ou nous écrivons ces lignes) le cas échéant.

3.1. Pour commencer

La première étape consiste à créer un projet adapté.

3.1.1. Xcode 3.0

Créez un nouveau projet par le menu File > New project…

Sélectionnez le type External Build System (Système de Compilation Externe)

Après avoir sélectionné l'emplacement du projet, Xcode vous fournit un projet vide prêt pour compiler un fichier Makefile.

3.1.2. Xcode 3.1 (avant la bêta 3 du SDK iPhone)

Créez un nouveau projet par le menu File > New project…

Sélectionnez le type de projet que vous souhaitez et nommez-le.

Dans la fenêtre de projet, supprimez toutes les entrées et les dossiers appartenant au projet.

Dans la section Targets, supprimez la cible par défaut.

3.1.3. Xcode 3.1 (après la bêta 3 du SDK iPhone)

Créez un nouveau projet par le menu File > New project…

Sélectionnez le type External Build System (Système de Compilation Externe) dans la section Others (Autres).

Après avoir sélectionné l'emplacement du projet, Xcode vous fournit un projet vide prêt pour compiler un fichier Makefile.

3.2. Préparer l'exécution de qmake

La deuxième étape consiste à préparer les cibles pour exécuter qmake puis compiler les Makefiles générés.

3.2.1. Xcode 3.0

Avec cette version de Xcode, vous disposez déjà d'une cible permettant l'appel de make et la compilation des Makefiles. Il ne vous reste plus qu'à ajouter la cible propre à qmake :

1. Rendez-vous dans le menu Project > New target… ;
2. Sélectionnez le type External Target (Cible externe) ;
3. Nommez la cible qmake et terminez. La cible doit apparaître dans la section Targets de votre projet ;

4. Double-cliquez sur cette nouvelle cible pour faire apparaître ses propriétés ;
5. Dans le champ Build Tool, entrez /usr/bin/qmake ;
6. Dans le champ Arguments, supprimez le texte existant et entrez –spec macx-g++ ;
7. Fermez la fenêtre.

3.2.2. Xcode 3.1

1. Rendez-vous dans le menu Project > New target… ;
2. Sélectionnez le type External Target (Cible externe) dans la section Others ;
3. Nommez la cible make et terminez. La cible doit apparaître dans la section Targets de votre projet ;
4. Une deuxième fois, rendez-vous dans le menu Project > New target… ;
5. Sélectionnez le type External Target (Cible externe) ;
6. Nommez la cible qmake et terminez. La cible doit apparaître dans la section Targets de votre projet ;
7. Double-cliquez sur cette nouvelle cible pour faire apparaître ses propriétés ;
8. Dans le champ Build Tool, entrez /usr/bin/qmake ;
9. Dans le champ Arguments, supprimez le texte existant et entrez –spec macx-g++ ;
10. Fermez la fenêtre.

3.3. Terminé !

Il ne nous reste plus qu'à déterminer les dépendances entre les cibles. Sélectionnez la cible qmake et glissez-la sur l'autre cible de votre projet pour que cette dernière en devienne dépendante. Si la dépendance a été validée, la cible qmake devrait apparaître comme un enfant de la première cible (cliquez éventuellement sur la flèche correspondante pour déplier la branche relative à la cible). Les dépendances ont une signification simple : toute cible qui dépend d'une autre exécutera la cible dont elle dépend avec elle-même. Dans le cas présent, nous demandons à ce que la cible active exécute la cible qmake pour générer les fichiers Makefiles à partir des données du projet avant de vouloir lancer la compilation avec make.

4. Fichiers .pro et début du développement

La dernière étape consiste à insérer dans le projet Xcode les éléments de votre projet Qt. Nous supposons que vos fichiers Qt se trouvent dans le même répertoire que le fichier .xcodeproj. Il vous suffit alors de les ajouter au projet Xcode. Pour ce faire, utilisez le menu Project > Add to Project… et sélectionnez votre fichier .pro, vos fichiers sources et éventuellement les autres fichiers susceptibles d'en faire partie (ressources, fichiers QtDesigner, etc.).

5. Compilation

À ce stade, votre projet devrait être prêt à être compiler. C'est l'heure de vérité. Tentez le menu Build > Build et contrôlez le résultat dans la barre d'état de Xcode ou dans la fenêtre Build > Build Results.

6. Lancement de l'exécutable

Il est temps maintenant d'exécuter votre logiciel. Utilisant un système de compilation externe, Xcode n'est pas en mesure de connaître de façon automatique l'emplacement de votre exécutable. Vous allez devoir l'ajouter au projet. Utilisez le menu Project > New Custom Executable… (Nouvel exécutable personnalisé) et sélectionnez le fichier de l'exécutable. Il devrait apparaître dans la section Executables du projet. En tant qu'unique exécutable du projet, il devient également l'exécutable actif et sera exécuté par défaut à chaque fois que vous souhaiterez exécuter votre projet.

Utilisez le menu Run > Run ou Run > Go pour le lancer.

7. Avantages et défauts

Le terme défauts est au pluriel dans ce titre mais la méthode exposée précédemment n'en présente qu'un seul : le fichier .pro n'est pas actualisé automatiquement en cas de nouveaux ajouts de fichiers à votre projet.

Certes, il s'agit là d'un manque à combler dans l'intégration de Qt à Xcode mais en comparaison des nombreux avantages que présente notre méthode, cela n'est pas grand chose :

- respect du standard instauré par Qt pour la compilation des projets ;
- portabilité du fichier .pro d'une plateforme à une autre ;
- aucune contrainte au moment du passage à un autre IDE que Xcode ;
- possibilité de modifier le projet Qt en dehors d'Xcode sans avoir à reconfigurer Xcode.

8. Conclusion

Comme toute chose, Xcode possède ses amateurs et ses détracteurs. En tant que produit Apple, il constitue un incontournable pour le développement d'applications natives Mac OS X. Bien que des environnements de développement portables spécifiques à Qt existent tels que MonkeyStudio, Edyuk ou encore QDevelop, les utilisateurs de Xcode trouveront peut-être un intérêt à éviter la multiplication des outils et nous espérons que ce tutoriel les aura aidé à améliorer leur entrée dans le monde de Qt.

Retrouvez l'article de Christophe Sauveur en ligne : Lien89

Développement Web pour mobiles - Les bases du HTML

Le langage HTML est le langage de base permettant de construire des pages Web, que celles-ci soient destinées à être affichées sur un iPhone / Android ou non. Dans notre cas, HTML sera associé à CSS et JavaScript pour construire des applications pour mobiles, qui seront accessibles sur le Web via une URL (applications Web), l'AppStore d'Apple ou l'Android Market (applications natives).

1. Préambule

HTML ou XHTML ? Ici c'est pareil !

Nous emploierons ici les termes HTML ou XHTML de façon identique. En effet, même si notre page HTML n'est pas formatée strictement comme l'exige XHTML, le navigateur Web sait l'interpréter de façon correcte et restituer l'affichage adéquat.

2. Notions générales

Une page HTML correspond à un fichier texte pouvant être écrit sous n'importe quel éditeur de texte, l'éditeur le plus simple d'utilisation étant dans ce cas le meilleur.

Le fichier texte contiendra principalement deux types de données :

- les balises HTML, qui permettent d'indiquer ce que l'on désire afficher (un paragraphe, une liste, une table, une image...) ;
- les attributs associés aux balises, qui permettent de préciser les paramètres de la balise (la taille de l'image, la couleur du texte...).

3. Balises

Une balise HTML est entourée des caractères < et >, par exemple pour indiquer un paragraphe. Une balise s'applique aux éléments qui la suivent. Pour indiquer la fin, on utilise la même balise, mais dont le nom est précédé de /.
Donc pour indiquer la fin du paragraphe, on écrira .

Un paragraphe en HTML
```
<p>Voici un paragraphe écrit en HTML</p>
```

Certaines balises ne possèdent pas de contenu. Par exemple, la balise
 permettant d'effectuer un retour à la ligne, n'a pas de contenu. Pour cela, elle s'écrira sous la forme suivante :

Une balise sans contenu
```
<br />
```

4. Attributs

Une balise utilisée sans attributs produira un fonctionnement standard. Par exemple, le paragraphe précédent aura la forme standard (police de caractères, couleur, *etc.*) associée à la balise employée ici. Si nous souhaitons avoir un paragraphe de couleur rouge (au lieu du noir standard), nous devons l'indiquer dans les attributs de la balise :

Un paragraphe de couleur rouge
```
<p style="color:red">Un paragraphe de couleur rouge</p>
```

Chaque balise a ses attributs propres, mais le plupart des balises possèdent les deux attributs style et class qui sont très utilisés pour définir des styles CSS (voir le prochain chapitre).

Nous avons ici utilisé l'attribut style, dont la valeur est "color:red". La forme générale d'écriture d'un attribut est nom="valeur". Des espaces peuvent figurer de part et d'autre du signe =. La valeur de l'attribut est généralement entourée de ' ou " (simple guillemet ou double guillemet). Ceux-ci peuvent être omis dans le cas où la valeur de l'attribut ne comporte pas d'espace. Dans ce cas, le prochain espace ou le indiquant la fin de la balise, sert au navigateur à connaître la valeur de l'attribut.

Dans le cas précédent, on aurait donc aussi pu écrire :

Pas de guillemets pour les valeurs d'attributs
```
<p style=color:red>Un paragraphe de couleur rouge</p>
```

Alors que ceci aurait été mal interprété :

Un espace dans la valeur d'un attribut (sans guillemets)
```
<p style=color: red>Un paragraphe de couleur rouge</p>
```

Dans ce cas, l'attribut style aurait la valeur color:, tandis que le mot red serait interprété comme le nom d'un nouvel attribut (qui n'existe évidemment pas !). Le paragraphe dans ce cas restera en couleur standard, c'est-à-dire noir.

Pour inclure un espace dans la valeur d'un attribut, il suffit de l'entourer de guillemets :

Un espace dans la valeur d'un attribut (avec guillemets)
```
<p style="color: red">Un paragraphe de couleur rouge</p>
```

Dans le cas où plusieurs attributs sont utilisés dans une balise, chaque couple nom="valeur" est séparé du suivant par au moins un espace, voire un retour à la ligne. L'ordre

dans lequel les attributs sont écrits n'a pas d'importance dans le résultat qui sera affiché.

5. Forme générale d'une page HTML

Une page HTML doit suivre quelques règles précises pour être correctement interprétée par le navigateur de l'utilisateur (dans notre cas le navigateur Safari de l'iPhone ou le navigateur Chrome d'un mobile Android).

Format standard d'une page HTML (fichier index.html)

```
<html>
  <head></head>
  <body>
    Voici une application pour iPhone !
  </body>
</html>
```

La page HTML indiquée ici correspond à un fichier texte dont le nom est par exemple index.html. L'extension du fichier permet de connaître le type de son contenu, ici du HTML !

Cette page HTML commence par la balise <html>, et se termine naturellement par cette même balise (</html>). Deux nouvelles balises sont introduites ici : <head> et <body>.

- La balise <head> correspond à la partie déclarative de la page : quels styles seront utilisés ? Quels fichiers de styles sont à inclure ? Quelle sera la taille maximum de l'écran ? Quel titre aura la page ? *etc.*
- La balise <body> correspond à ce qui sera affiché à l'utilisateur : le texte, les images, les listes, *etc.*

6. Visualisation d'une page HTML

La page HTML écrite précédemment peut être visualisée dans n'importe quel navigateur actuel : Internet Explorer, Firefox, Google Chrome, ou encore Safari.

Si vous souhaitez visualiser cette page depuis votre ordinateur personnel (PC, Mac...), il suffit de faire glisser le fichier dans votre navigateur. Pour l'afficher sur votre iPhone / Android, il faut d'abord installer un serveur Web sur votre ordinateur personnel, par exemple un serveur PHP (ou tout autre serveur). Dans cet ouvrage nous avons utilisé deux types de serveurs :

- AppServ sous Windows permettant d'afficher des pages écrites en PHP (et bien sûr aussi HTML) ;
- MAMP sous Mac permettant d'afficher des pages écrites en PHP (et bien sûr aussi HTML) ;
- Ruby on Rails (Windows ou Mac) permettant d'afficher des pages écrites en Ruby (et également en HTML).

Des serveurs Java, Microsoft .Net, *etc.*, fonctionneraient de manière similaire.

Visualisons notre page HTML en ouvrant, par exemple, Safari sur un iPhone et en tapant l'adresse de notre serveur local dans la barre d'adresse :

Une page HTML affichée sur l'iPhone

Le texte inclus dans la balise <body> s'affiche, mais dans une taille très petite et pas du tout adaptée à l'iPhone ! Ceci car notre page HTML, sauf indication contraire, est écrite pour être affichée dans un navigateur d'un ordinateur de bureau, et pas pour le navigateur d'un téléphone portable !

Il suffit d'ajouter la ligne suivante dans la partie <head> du code HTML pour afficher correctement la page HTML :

Avoir la taille de la page adaptée à la taille de l'écran

```
<meta name="viewport" content="user-
scalable=no,width=device-width" />
```

Une fois cette balise <meta> insérée dans la partie <head>, l'affichage est maintenant :

Adaptation de l'affichage à la taille de l'écran

Par la suite, nous conserverons cette balise <meta> dans la partie <head> de chacune des pages HTML que nous écrirons.

ASTUCE : et pour Android, utile ou pas ? Android n'a peut être pas besoin de cette balise <meta>. Toutefois,

nous l'inclurons dans toutes nos sources de façon à être compatible pour iPhone et Android.

7. Le texte

Nous avons précédemment utilisé une balise <p> pour afficher un paragraphe de texte. D'autres balises existent permettant d'afficher du texte sous différentes formes :

- <p> : affiche un paragraphe. Deux paragraphes qui se suivent seront donc espacés verticalement à l'écran ;
- <b> : met le texte en gras (bold) ;
- <i> : met le texte en italique ;
-
 : insère un retour à la ligne. Cette balise ne possède pas de contenu, d'où sa forme d'écriture ;
- <hr /> : insère une ligne de séparation horizontale. Cette balise ne possède également pas de contenu ;
- <h1> : affiche un titre de paragraphe. Pour différencier différents niveaux de titres, on utilise les autres balises <h2>, <h3>, <h4>, <h5> et <h6>. Plus l'indice qui suit **h** augmente, plus le titre est petit. L'indice qui suit **h** symbolise donc l'imbrication ou le niveau du titre.

En plus des balises décrites ci-dessus, il existe des codes spéciaux permettant d'écrire certains caractères. Ces codes spéciaux s'appellent des entités HTML. En voici quelques-unes parmi les plus utilisées :

-   : insère un espace dit insécable entre deux mots. En effet, le navigateur ne tient pas compte du formatage du texte que nous tapons dans notre fichier HTML. Les retours à la ligne effectués par l'appui sur la touche **Return** ne sont pas pris en compte (lors de l'affichage), pas plus que les espaces insérés entre les mots (au-delà du premier espace). On a vu que la balise
 permettait d'insérer un retour à la ligne. L'entité   permet d'insérer un espace supplémentaire entre deux mots ;
- < : le caractère < est un caractère réservé en HTML, signifiant le début ou la fin d'une balise. Pour afficher le signe < à l'écran, on utilise donc l'entité < ;
- > : idem que <, mais pour afficher le caractère > (qui sinon serait interprété comme la fin de la balise).

Voici un petit exemple contenant quelques balises HTML :

Quelques balises HTML

```
<html>
  <head>
    <meta name="viewport" content="user-
scalable=no,width=device-width" />
  </head>
  <body>
    <h1>Mode d'emploi des balises</h1>
    <p>Commençons par le plus <b>important</b> :
</p>
    <h2>En gras</h2>
    <b>Ceci est en gras</b>
    <h2>En italique</h2>
    <i>Ceci est en italique</i>
    <h2>En gras et en italique</h2>
    <b><i>Ceci est en gras et en italique</i></b>
  </body>
</html>
```

On peut voir l'affichage produit :

Utilisation de balises dans la page

8. Les images

Les images s'affichent par l'intermédiaire de la balise <img>. Celle-ci ne possède pas de contenu, donc on l'utilise sous la forme <img />.

Des attributs sont nécessaires pour afficher l'image :

- src : permet d'indiquer l'adresse Internet où se trouve l'image. Cet attribut est obligatoire (sinon on ne peut pas afficher l'image !).

  Si l'image se trouve sur le serveur (ce qui est souvent le cas), l'adresse peut être relative à la page HTML qui affiche l'image. On n'indique donc pas dans ce cas, le chemin complet d'accès à l'image (commençant par http://), mais le chemin relatif (par exemple images/img1.png si l'image se trouve dans un répertoire images situé au même niveau que la page HTML).

  Si l'image n'est pas sur le serveur, le chemin complet commençant par http:// sera nécessaire ;

- width : cet attribut permet d'indiquer une largeur à l'image. Si l'image est plus petite que la valeur indiquée, l'image est agrandie, sinon elle est rétrécie ;
- height : idem que width, mais pour la hauteur de l'image.

Par exemple, pour afficher l'image img1.png sur une hauteur de 200 pixels :

Une image de 200 pixels de hauteur

```
<img src=img1.png height=200 />
```

On remarque que l'image est redimensionnée en proportion de la hauteur indiquée, sauf si l'attribut width est également indiqué.

ASTUCE : pour ne pas agrandir trop ? max-height et max-width !
Les attributs width et height peuvent être remplacés par les propriétés CSS max-width et max-height qui permettent de ne pas agrandir l'image si la taille est plus petite que celle indiquée (voir chapitre suivant).

9. Les liens

Un lien correspond à l'utilisation de la balise <a>. Le lien permet de naviguer vers une autre page HTML, voire la même. Pour cela on utilise l'attribut href dans la balise <a> :

- si href commence par #, cela signifie que l'on désire atteindre une partie de la même page HTML. Le nom qui suit href (par exemple href=#partie1) correspond à un endroit du code dans la page HTML identifié par ce nom. Ce mécanisme sera utilisé par la bibliothèque iUI que nous étudierons dans la partie suivante ;
- sinon (href est indiqué mais ne commence pas par #), cela référence une autre page HTML. Lorsque le lien est cliqué, la page est affichée.

Des valeurs particulières de href sont possibles, permettant des actions spéciales :

- mailto:une_adresse_email : permet de passer en mode d'écriture d'un nouveau mail qui sera adressé à l'adresse email indiquée ;
- tel:un_numero_de_telephone : propose de téléphoner au contact indiqué par le téléphone ;
- sms:un_numero_de_telephone : idem que tel, mais pour envoyer un sms.

COMPATIBILITÉ : et avec les sites Web classiques ?

Les valeurs tel et sms dans l'attribut href ne sont disponibles que sur iPhone / Android, alors que mailto peut aussi s'utiliser dans des pages Web classiques.

Voici un code HTML permettant d'envoyer un mail, de téléphoner et d'envoyer un sms.

Envoyer un mail, un sms ou téléphoner.

```
<html>
  <head>
        <meta name="viewport" content="user-
scalable=no,
                width=device-width" />
  </head>

  <body>
    <a href=mailto:ericsarrion@gmail.com >
          Envoyer un mail </a><br /><br />
    <a href=tel:0612345678 >Téléphoner </a><br
/><br />
    <a href=sms:0612345678 >Envoyer un sms </a>
  </body>
</html>
```

Lors de l'exécution de cette page HTML, on peut remarquer que le clic sur le lien sms ouvre une nouvelle fenêtre, mais ne retourne pas à notre application après l'envoi du sms, au contraire des liens mailto et tel qui permettent le retour à celle-ci.

10. Les listes

Les listes sont un élément essentiel pour afficher des informations sur l'iPhone / Android. En effet, de part leur petit affichage, il va être nécessaire de présenter la plupart des informations sous forme de liste. Un élément de liste prendra en général la largeur de l'affichage, tandis que les éléments de liste seront placés les uns sous les autres.

HTML permet d'afficher les listes au moyen des balises <ul> et <li>. Il existe également la balise <ol>, mais celle-ci ne sera pas utilisée dans cet ouvrage.

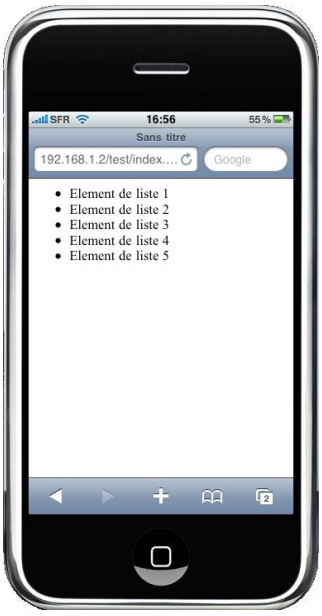
- : permet d'indiquer le début d'une liste, qui se terminera donc par . signifie *unordered list*, ce qui se traduit par liste non ordonnée, au contraire de <ol> qui signifie *ordered list* (liste ordonnée).
- : permet d'indiquer un élément dans la liste, qui correspondra à une ligne de l'affichage. Cette ligne pourra contenir d'autres éléments HTML (une image, un texte, un lien...).

Voici une liste composée de cinq éléments :

Liste d'éléments

```
<html>
  <head>
    <meta name="viewport" content="user-
scalable=no, width=device-width" />
  </head>
  <body>
    <ul>
      <li> Element de liste 1 </li>
      <li> Element de liste 2 </li>
      <li> Element de liste 3 </li>
      <li> Element de liste 4 </li>
     <li> Element de liste 5 </li>
    </ul>
  </body>
</html>
```

Elle s'affiche comme ceci dans l'iPhone :



Affichage d'éléments de liste

L'affichage est dans le style d'une page HTML tel qu'on le rencontre sur la plupart des sites Internet, mais n'est pas tellement dans le style iPhone ou Android ! Nous verrons que l'utilisation des styles CSS permet d'obtenir un meilleur affichage.

11. Les blocs

Comme les listes, les blocs sont une composante essentielle de l'affichage sur iPhone / Android. Ils correspondent aux éléments et de HTML :

- : permet d'afficher un bloc qui ne pourra être mitoyen que de l'élément situé au dessus et / ou au-dessous, mais n'aura pas de mitoyenneté sur les côtés droit ou gauche ;
- : à l'inverse du , il pourra avoir une mitoyenneté sur les côtés droit et / ou gauche, en plus du haut et du bas de l'élément.

On voit donc que les éléments ne pourront s'afficher que les uns sous les autres, tandis que les éléments pourront s'afficher les uns à côté des autres.

De plus, un élément peut contenir d'autres éléments ou (qui s'afficheront selon les mêmes règles), tandis qu'un élément ne pourra pas contenir d'éléments (un est censé être un bloc occupant toute la ligne, donc il est difficile d'en mettre deux ou plus par ligne...).

Voici un exemple de code HTML contenant ces deux éléments :

Elements div et span

```
<html>
  <head>
    <meta name="viewport" content="user-
scalable=no, width=device-width" />
  </head>
  <body>
    <div>Un premier div</div>
    <div>Un second div</div>
    <div>
      <span>Du texte</span>
      <span>contigu à celui-ci</span>
    </div>
  </body>
</html>
```

Cela s'affiche de la façon suivante :

Utilisation de div et span

Les éléments se mettent bien les uns sous les autres, tandis que les éléments se mettent les uns à côté des autres. Pour que deux éléments soient l'un en dessous de l'autre, il faut soit les mettre dans deux éléments différents, soit casser la séquence en insérant, par exemple, un élément
 entre les deux.

12. La suite ?

Vous avez aimé cet article et vous voulez avoir la suite de l'histoire ? Elle est dans mon livre XHTML / CSS et JavaScript pour le Web mobile : Lien90.

Retrouvez l'article de Eric Sarrion en ligne : Lien91

Mac OS 10.6 Snow Leopard : les fondamentaux

Cette formation vidéo regroupe tout ce qu'il faut savoir pour démarrer rapidement et simplement avec Mac OS X Snow Leopard. Si vous avez déjà travaillé avec une version antérieure à Mac OS X, le premier chapitre vous propose une vue d'ensemble de toutes les nouveautés essentielles.

Le deuxième chapitre est plus particulièrement destiné aux débutants. Vous commencerez avec l'interface utilisateur de Mac OS X. Vous serez alors surpris de voir à quel point il est facile et pratique d'effectuer n'importe quelle tâche. Afin que vous puissiez exploiter tout le potentiel du système, votre formateur Jeremy Veron vous présente en détail tous ses avantages.

Puis vous découvrirez les différentes technologies du système d'exploitation Mac OS X que ce soit au niveau hardware ou software, **Spotlight** pour effectuer des recherches, **Time Machine** pour gérer les sauvegardes, ou bien encore **QuickTime X** pour la présentation de vidéos et de supports multimédia.

Les chapitres suivants vous montreront comment votre Mac interagit avec son environnement local, avec les différents utilisateurs, les différentes imprimantes, les connexions réseaux, Internet, etc. Vous apprendrez également à utiliser le calendrier, la calculatrice, le répertoire d'adresses, le traitement de texte, la création de fichiers PDF, la gestion des caractères ou encore avec la lecture de DVD et de musique.

Pour les utilisateurs les plus avertis : des ateliers consacrés aux Scripts, aux raccourcis clavier, aux méthodes de travail, ainsi qu'à la structure originelle UNIX qui est le cœur du système Mac OS X. Vous saurez tout ce qui se cache à l'intérieur !

Critique du livre par Marcos Ickx

On dit toujours qu'une image vaut mieux qu'un long discours.

J'ai envie de dire qu'une vidéo vaut mieux qu'un épais livre.

Quoi de plus agréable en effet, que de voir à l'écran toutes les étapes effectuées par la personne pour réaliser une action bien précise.

Ce qui aurait pris plusieurs pages dans un livre pour être clairement expliqué l'est ici encore plus clairement en quelques secondes.

En effet, cette formation nous propose toute une série de vidéos qui varient en longueur, allant de 4 minutes pour la plus courte à 22 minutes pour la plus longue.

Chacune de ces vidéos couvre une fonctionnalité bien précise de Mac OS X Snow Leopard, et revient, lorsque c'est nécessaire sur des fonctionnalités dont on n'aurait même pas soupçonné l'existence (comme l'affichage et la manipulation d'objets 3D, par exemple, ou le détourage d'une image sans avoir à installer une application supplémentaire).

Vraiment, j'ai été surpris de découvrir que derrière certains outils apparemment simplistes (Aperçu, par exemple), se cachaient en fait de puissantes fonctionnalités, mais utilisables très simplement.

Les vidéos sont vraiment très agréables à regarder, même lorsqu'on les affiche en plein écran (la résolution de mon écran est de 1680*1050) et l'on comprend très bien le formateur.

Deux petites choses qui m'ont gênées, mais qui n'ont rien à voir avec le contenu des vidéos, contenu qui est de très bonne qualité :

- on ne sait pas quelles sont les vidéos qu'on a déjà visionnées et celles qu'on n'a pas encore vues : il n'y a pas moyen de marquer une vidéo comme étant vue ou de mettre un signet là où on s'est arrêté ;
- le volume sonore varie d'une vidéo à l'autre. il faut donc l'ajuster à chacune des vidéos.

Pour le reste, j'ai trouvé la partie consacrée à Unix et au terminal quelque peu légère.
J'aurais aimé que le formateur aille un peu plus loin dans cette partie.

Mais il ne faut pas oublier qu'il s'adresse à des débutants.

Retrouvez cette critique de livre sur la page livres Mac : Lien92

Stuxnet renfermerait une allusion biblique - Des experts suspectent fortement Israël d'être l'auteur du ver

Jusqu'ici, le très complexe ver Stuxnet donnait plus dans le registre des films d'espionnage.
Son but principal étant, visiblement, de pénétrer les infrastructures industrielles iraniennes (et chinoises).

Mais avec la « découverte » de plusieurs experts en sécurité, Stuxnet plonge également en plein Da Vinci Code.

D'après les experts en question, le code du ver comporterait un fichier nommé Myrtus (en français : l'arbre de Myrte). Un nom très inhabituel et surtout hautement symbolique dans le culture biblique.

Symbole de paix et d'espoir de jours meilleurs (*« Au lieu du buisson croîtra le sapin et au lieu de l'épine croîtra le Myrte ; et cela rendra glorieux le nom de l'Éternel et sera un signe perpétuel, qui ne sera jamais retranché »*), le Myrte est aussi symbole de justice dans l'Ancien Testament.

C'est sur cette dernière interprétation que s'arrêtent les tenants de cette thèse de la référence biblique. Une thèse rapportée dans le très sérieux New York Times. Pour eux, ce nom de fichier serait ouvertement une allusion au Livre d'Esther (et donc à la Torah).

Livre d'Esther où il y est dit : *« Elle s'appelait Hadassah parce qu'elle était juste et que l'on compare au Myrte ceux qui aiment la justice»*. Hadassah est l'un des noms de la reine Esther et signifie en hébreux : Myrte.

Or ce Livre raconte comment, sous la direction de la reine Hadassah donc, le peuple juif déjoua des attaques perses destinées à l'anéantir.

On voit immédiatement l'auteur du ver désigné par cette thèse.

Farfelu ? Tiré par les cheveux ? Réel indice de la provenance du ver ?

Les supputations vont bon train. Toujours est-il que ce fichier est bel et bien présent dans le code. Mais il pourrait aussi bien *« faire partie d'un jeu d'esprit [que] montrer la négligence ou bien la fantaisie des codeurs »* constate le journal.

Des développeurs israéliens auraient-ils eu la négligence de laisser de tels indices derrière eux ? Ou s'agit-il d'une diversion pré-étudiée pour diriger les soupçons vers un pays dont les services secrets sont bien connus ?

Les preuves sont légères pour répondre à ces questions, voire inexistantes.

De son côté, et sans surprise, le gouvernement israélien dément toute implication dans cette cyber-attaque.

Sur le ver en lui-même on sait toujours assez peu de choses. On suppose que la première version du ver est apparue en 2009, puis qu'elle a été modifiée début 2010. Par qui ? Encore une question.

L'origine de Stuxnet ? Mystère également.

Et l'on risque de ne pas en savoir beaucoup plus. *« Lors d'entretiens dans plusieurs pays, les experts en cyber-guerre et en technologie nucléaire disent que le mystère Stuxnet pourrait ne jamais être résolu »*.

Un rébus enveloppé de mystère au sein d'une énigme, pourrait-on dire pour paraphraser Winston Churchill, experts parmi les experts des relations internationales et de l'espionnage.

Commentez cette news de Katleen Erna en ligne : Lien93

Le réseau social open-source Diaspora libère son code ce jour, avant un lancement au cours du mois d'octobre

Il y a quelques mois, nous vous parlions du projet Diaspora : un réseau social open-source, qui souhaitait proposer une alternative à Facebook. Depuis, quelques 200.000 dollars ont été récoltés pour son développement.

Les conditions de son lancement restaient un peu floues. Elles ont aujourd'hui été dévoilées par les créateurs du site, qui entrera en service pour tous (en version alpha) courant octobre.

Mais son code, qui est libéré ce jour, est bien avancé et Diaspora serait stable. Ses quatre co-fondateurs le décrivent comme « un réseau distribué, dans lequel des ordinateurs séparés se connectent directement les uns aux autres, ce qui permet de se connecter sans mettre à mal sa vie privée ».

Pour eux, les machines sont des « graines » et seront en possession de l'utilisateur, qui les hébergera chez lui ou sur un serveur dédié (une version payante hébergée par les développeurs sera proposée sans publicité).

Une fois configurée, la « graine » récoltera vos informations : profil Facebook, tweets, etc.

Mais avec un seul mot d'ordre : un partage « clair et

contextuel ». C'est-à-dire que les données qui seront partagées le seront clairement et avec différentes catégories de personnes (les amis, les proches, tout le Net, etc.). Une manière subtile de faire un pied de nez à Facebook et à ses paramètres de confidentialité qui ont fait couler beaucoup d'encre.

Les étudiants derrière le projet affirment cependant que son lancement ne sera qu'un début : « cette première sortie sera le départ d'un grand projet sur le long terme, et non pas l'achèvement d'un simple projet d'été », ont-ils conclu. À voir.

Commentez cette news de Katleen Erna en ligne : Lien94

Les derniers tutoriels et articles

Présentation des privilèges d'exécution dans l'environnement Microsoft Windows

Cet article a pour but de présenter la notion de "privilèges" dans l'environnement Microsoft Windows et de montrer par un exemple simple écrit en C leur manipulation.

1. Introduction

La connexion d'un utilisateur est un processus complexe dans le monde Windows. Mais au bout du compte et quelle que soit la méthode d'authentification utilisée (local, domaine, réseau, Terminal Service…), l'utilisateur se voit attribuer un jeton d'authentification (appelé aussi token dans la littérature anglaise).

Ce jeton contient plusieurs informations :

- l'identifiant de sécurité de l'utilisateur appelé aussi Secury IDentifier ou SID dans la littérature anglaise ;
- le ou les SID des groupes auxquels appartient cet utilisateur ;
- un SID de session (logon SID) qui sert à identifier la session de l'utilisateur ;
- la liste des privilèges de l'utilisateur ;
- l'identification de la source du jeton. Cette identification permet de reconnaître le type de session : Session Manager, LAN Manager, RPC Server...
- une variable permettant de savoir s'il s'agit d'un jeton principal (primary Token) ou d'un jeton modifié (Impersonation Token) ;
- et beaucoup d'autres choses encore.

Cet article s'intéresse exclusivement aux privilèges Windows donnés par le système et inscrits dans ce jeton d'authentification.

Cet article est plus particulièrement destiné aux administrateurs de systèmes Windows, aux développeurs d'applications orientées "système" et aussi aux personnes curieuses de connaitre un peu mieux le fonctionnement interne de Windows.

2. Les privilèges

2.1. Droit ou privilège ?

Il est facile de faire la confusion entre ces deux mots, une clarification est donc nécessaire.

Un **droit** est appliqué à un objet d'une arborescence (fichier, répertoire, clé de registre...) pour un ou plusieurs utilisateurs ou un ou plusieurs groupes d'utilisateurs. Ainsi on parlera du droit de lecture, d'écriture, d'exécution, de parcours et d'autres droits encore liés à la gestion d'une arborescence.

Un **privilège** s'applique à un ou plusieurs utilisateurs ou à un ou plusieurs groupes d'utilisateurs et il concerne une opération spécifique possible sur le noyau ou le système. Ainsi, on parlera du privilège d'arrêter le système, du privilège de modifier l'heure du système...

2.2. Attribution et portée des privilèges

Au niveau du système, il existe une base de données, la base LSA (Local Security Authority), qui contient, entre autres, la liste des privilèges attribués à chaque utilisateur ou groupe d'utilisateurs.

Les privilèges s'appliquent donc au niveau local et ne sont pas transmis au travers du réseau vers une autre machine. Il est ainsi tout à fait possible et normal d'avoir certains privilèges sur une machine et d'autres privilèges sur d'autres machines avec le même nom d'utilisateur.

La plupart des privilèges, même s'ils sont attribués à un utilisateur, sont désactivés par défaut. C'est au programme qui souhaite les utiliser de les activer si nécessaire.

Quand le système tente d'effectuer une opération qui nécessite certains privilèges, il vérifie que l'utilisateur possède bien ces privilèges et si oui, il vérifie que ces privilèges sont bien activés. Si l'un de ces deux tests échoue, le système refuse d'effectuer l'opération demandée.

Au niveau du noyau, chaque privilège est associé à un LUID (Local Unique IDentifier). Cet identifiant, comme son nom l'indique, est local et unique pour chacune des machines (il semblerait même que cet identifiant de privilège soit différent d'une connexion à l'autre). C'est aussi pour cette raison qu'un privilège ne peut être transmis à une autre machine car au niveau du noyau, la seule chose qui est associée au jeton d'identification d'un utilisateur, c'est une liste de LUID de privilèges et pas une liste de noms de privilèges.

2.3. La liste des privilèges

Sous Windows, la liste des privilèges disponibles se trouve dans le tableau ci-dessous. Ce tableau a été établi à l'aide de MSDN "Account Rights Constants" (Lien95) et "Privilege Constants" (Lien96) et aussi l'aide en ligne disponible sur un système Windows Seven Professionnal.

Ce tableau contient quatre colonnes :

- la colonne 1 "Nom/Constante" est l'identifiant du privilège (tel que défini dans le fichier winnt.h) et la chaîne de caractères identifiant le privilège. Un développeur ne doit utiliser que l'identifiant du privilège, pas la chaîne de caractères ;
- la colonne 2 "Description" est une description courte du privilège ;
- la colonne 3 "Explication" contient des informations plus détaillées sur ce privilège ;
- la colonne 4 "Valeur par défaut" enfin, donne les valeurs par défaut, c'est-à-dire les groupes à qui sont normalement attribués ces privilèges. Certaines valeurs peuvent être différentes suivant que la machine est une station de travail (identifié par WKS), un serveur (identifié par SRV) ou un contrôleur de domaine (identifié par DC).

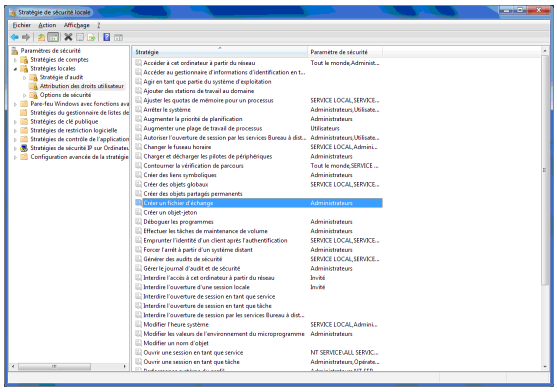
Voir le tableau en ligne : Lien97

3. La gestion des privilèges

3.1. Gestion des privilèges graphique

La gestion des privilèges se fait à l'aide d'un plug-in lancé par une MMC (**M**icrosoft **M**anagement **C**onsole) se trouvant dans le panneau de configuration, dans les outils d'administration et qui s'appelle "Stratégie de sécurité locale" (du moins sur un système Windows Seven Professionnel français).

Ce plug-in peut aussi être lancé directement par la ligne de commande suivante : **"%windir %\system32\secpol.msc /s"**

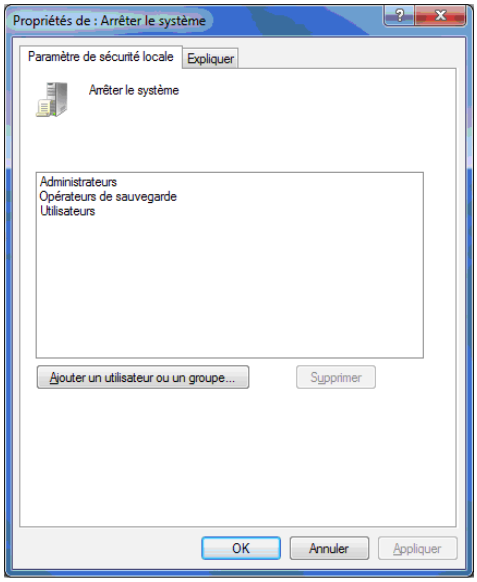


Le gestionnaire des privilèges secpol.msc

Il semble que ce plug-in n'existe pas par défaut sur les versions familiales de Microsoft Windows.

La gestion des privilèges se trouve dans la section "Stratégies locales" / "Attribution des droits d'utilisateur" de la partie gauche de l'arbre.

Il est possible de modifier l'affectation des privilèges pour un utilisateur ou un groupe d'utilisateurs. Pour cela, il suffit de sélectionner un des privilèges affichés et ensuite, de faire un clic droit et de sélectionner l'option "Propriétés".



Les propriétés du privilège 'Arrêter le système'

3.2. Gestion des privilèges à la console

La gestion des privilèges peut aussi se faire à la console. Il suffit pour cela d'installer l'outil **NtRights.exe** qui fait partie du "**Windows Server 2003 Resource Kit Tools**". Ce "Resource Kit Tools" peut être téléchargé ici : Lien98.

4. Un petit programme de démonstration

4.1. But du programme

Le programme présenté dans cet article est relativement simple, il se propose :

- d'inventorier les privilèges détenus par l'utilisateur courant ;
- de tenter de prendre un privilège non détenu par le jeton d'authentification de l'utilisateur (le privilège SE_TCB_NAME) ;
- d'activer un des privilèges détenus par le jeton d'authentification de l'utilisateur (le privilège SE_SHUTDOWN_NAME) ;
- d'inventorier à nouveau les privilèges détenus par l'utilisateur courant.

4.2. Environnement de développement

Le programme de test a été développé avec l'environnement suivant :

- Microsoft Windows Seven Profesionnel avec tous les correctifs de sécurité appliqués au jour de la rédaction de l'article (15 septembre 2009) ;
- Visual Studio 2005 ;
- programme en mode console compilé en Unicode ;
- la gestion des erreurs est ultra-simpliste ;
- le programme est écrit volontairement en C afin que d'autres langages puissent utiliser les idées développées dans ce code.

Pour les plus impatients, le binaire peut être téléchargé ici : Lien99
et le projet complet (au format Visual Studio 2005) peut être téléchargé ici : Lien100.

Si une erreur se produit lors du lancement du programme **Privileges.exe**, il faut installer les **redistribuables VS 2005**. Ce setup peut être téléchargé ici : Lien101.

4.3. Les fonctions utilisées

Ce paragraphe liste les fonctions spécifiques à la gestion des privilèges Microsoft et utilisées par le programme de démonstration :

- la fonction OpenProcessToken() permet d'obtenir un jeton sur un processus quelconque. Ce jeton contient, entre autres, toutes les informations concernant les privilèges détenus et leur état : Lien102 ;
- la fonction GetTokenInformation() permet de récupérer un ensemble d'informations concernant le jeton passé en paramètre. Parmi ces informations, on peut récupérer la liste des privilèges ainsi que leur état actif ou non : Lien103 ;
- la fonction LookupPrivilegeValue() permet de récupérer le Local Unique Identifier (LUID) associé à un privilège dont on spécifie le nom : Lien104 ;
- la fonction LookupPrivilegeName() permet de récupérer le nom d'un privilège associé à un Local Unique Identifier (LUID) : Lien105 ;
- la fonction LookupPrivilegeDisplayName() permet de récupérer le nom affiché pour un privilège donné. C'est ce nom qui est utilisé dans la console de gestion des privilèges secpol.msc : Lien106 ;
- la fonction AdjustTokenPrivileges() permet de modifier l'état (activé ou désactivé) des privilèges possédés par le jeton passé en paramètre : Lien107 ;
- la fonction EnumeratePrivilegeName() : cette fonction n'existe pas, Microsoft n'a rien prévu dans son API pour énumérer tous les privilèges connus et disponibles.

4.4. Le résultat généré

Le résultat de l'exécution de la version Release du programme de démonstration sur un poste Windows Seven professionnel produit le résultat suivant :

Résultat du programme de démonstration

```
D:\Raymond\Developpement\Programmes\Privileges\Release>Privileges.exe
Affichage des privilèges d'un jeton
"SeShutdownPrivilege" (Arrêter le système)
        Privilège désactivé
"SeChangeNotifyPrivilege" (Contourner la vérification de parcours)
        Privilège activé par défaut
        Privilège activé
"SeUndockPrivilege" (Retirer l'ordinateur de la station d'accueil)
        Privilège désactivé
"SeIncreaseWorkingSetPrivilege" (Augmenter une plage de travail de processus)
        Privilège désactivé
"SeTimeZonePrivilege" (Changer le fuseau horaire)
        Privilège désactivé

Tentative de prise du privilège SeTcbPrivilege
        Erreur AdjustTokenPrivileges() : L'appelant ne bénéficie pas de tous les privilèges ou groupes référencés.

Tentative de prise du privilège SeShutdownPrivilege
        ==> OK

Affichage des privilèges d'un jeton
"SeShutdownPrivilege" (Arrêter le système)
        Privilège activé
"SeChangeNotifyPrivilege" (Contourner la vérification de parcours)
        Privilège activé par défaut
        Privilège activé
"SeUndockPrivilege" (Retirer l'ordinateur de la station d'accueil)
        Privilège désactivé
"SeIncreaseWorkingSetPrivilege" (Augmenter une plage de travail de processus)
        Privilège désactivé
"SeTimeZonePrivilege" (Changer le fuseau horaire)
        Privilège désactivé

D:\Raymond\Developpement\Programmes\Privileges\Release>
```

Le même programme lancé dans un shell avec les privilèges d'administrateur donne beaucoup plus de privilèges (et c'est normal) :

Résultat du programme de démonstration en mode Administrateur

```
D:\Raymond\Developpement\Programmes\Privileges\Release>Privileges.exe
Affichage des privilèges d'un jeton
"SeIncreaseQuotaPrivilege" (Ajuster les quotas de mémoire pour un processus)
        Privilège désactivé
"SeSecurityPrivilege" (Gérer le journal d'audit et de sécurité)
        Privilège désactivé
"SeTakeOwnershipPrivilege" (Prendre possession de fichiers ou d'autres objets)
        Privilège désactivé
"SeLoadDriverPrivilege" (Charger et décharger les pilotes de périphériques)
        Privilège désactivé
"SeSystemProfilePrivilege" (Performance système du profil)
        Privilège désactivé
"SeSystemtimePrivilege" (Modifier l'heure système)
        Privilège désactivé
"SeProfileSingleProcessPrivilege" (Processus unique du profil)
        Privilège désactivé
"SeIncreaseBasePriorityPrivilege" (Augmenter la priorité de planification)
        Privilège désactivé
"SeCreatePagefilePrivilege" (Créer un fichier d'échange)
        Privilège désactivé
"SeBackupPrivilege" (Sauvegarder les fichiers et les répertoires)
        Privilège désactivé
"SeRestorePrivilege" (Restaurer les fichiers et les répertoires)
        Privilège désactivé
"SeShutdownPrivilege" (Arrêter le système)
        Privilège désactivé
```

```
"SeDebugPrivilege" (Déboguer les programmes)
        Privilège désactivé
"SeSystemEnvironmentPrivilege" (Modifier les
valeurs de l'environnement du microprogramme)
        Privilège désactivé
"SeChangeNotifyPrivilege" (Contourner la
vérification de parcours)
        Privilège activé par défaut
        Privilège activé
"SeRemoteShutdownPrivilege" (Forcer l'arrêt à
partir d'un système distant)
        Privilège désactivé
"SeUndockPrivilege" (Retirer l'ordinateur de la
station d'accueil)
        Privilège désactivé
"SeManageVolumePrivilege" (Effectuer les tâches
de maintenance de volume)
        Privilège désactivé
"SeImpersonatePrivilege" (Emprunter l'identité
d'un client après l'authentification)
        Privilège activé par défaut
        Privilège activé
"SeCreateGlobalPrivilege" (Créer des objets
globaux)
        Privilège activé par défaut
        Privilège activé
"SeIncreaseWorkingSetPrivilege" (Augmenter une
plage de travail de processus)
        Privilège désactivé
"SeTimeZonePrivilege" (Changer le fuseau horaire)
        Privilège désactivé
"SeCreateSymbolicLinkPrivilege" (Créer des liens
symboliques)
        Privilège désactivé

Tentative de prise du privilège SeTcbPrivilege
        Erreur AdjustTokenPrivileges() :
L'appelant ne bénéficie pas de tous les
privilèges ou groupes référencés.

Tentative de prise du privilège
SeShutdownPrivilege
        ==> OK

Affichage des privilèges d'un jeton
"SeIncreaseQuotaPrivilege" (Ajuster les quotas de
mémoire pour un processus)
        Privilège désactivé
"SeSecurityPrivilege" (Gérer le journal d'audit
et de sécurité)
        Privilège désactivé
"SeTakeOwnershipPrivilege" (Prendre possession de
fichiers ou d'autres objets)
        Privilège désactivé
"SeLoadDriverPrivilege" (Charger et décharger les
pilotes de périphériques)
        Privilège désactivé
"SeSystemProfilePrivilege" (Performance système
du profil)
        Privilège désactivé
"SeSystemtimePrivilege" (Modifier l'heure
système)
        Privilège désactivé
"SeProfileSingleProcessPrivilege" (Processus
unique du profil)
        Privilège désactivé
"SeIncreaseBasePriorityPrivilege" (Augmenter la
priorité de planification)
        Privilège désactivé
"SeCreatePagefilePrivilege" (Créer un fichier
d'échange)
        Privilège désactivé
"SeBackupPrivilege" (Sauvegarder les fichiers et
les répertoires)
        Privilège désactivé
"SeRestorePrivilege" (Restaurer les fichiers et
les répertoires)
        Privilège désactivé
"SeShutdownPrivilege" (Arrêter le système)
        Privilège activé
"SeDebugPrivilege" (Déboguer les programmes)
        Privilège désactivé
"SeSystemEnvironmentPrivilege" (Modifier les
valeurs de l'environnement du microprogramme)
        Privilège désactivé
"SeChangeNotifyPrivilege" (Contourner la
vérification de parcours)
        Privilège activé par défaut
        Privilège activé
"SeRemoteShutdownPrivilege" (Forcer l'arrêt à
partir d'un système distant)
        Privilège désactivé
"SeUndockPrivilege" (Retirer l'ordinateur de la
station d'accueil)
        Privilège désactivé
"SeManageVolumePrivilege" (Effectuer les tâches
de maintenance de volume)
        Privilège désactivé
"SeImpersonatePrivilege" (Emprunter l'identité
d'un client après l'authentification)
        Privilège activé par défaut
        Privilège activé
"SeCreateGlobalPrivilege" (Créer des objets
globaux)
        Privilège activé par défaut
        Privilège activé
"SeIncreaseWorkingSetPrivilege" (Augmenter une
plage de travail de processus)
        Privilège désactivé
"SeTimeZonePrivilege" (Changer le fuseau horaire)
        Privilège désactivé
"SeCreateSymbolicLinkPrivilege" (Créer des liens
symboliques)
        Privilège désactivé

D:\Raymond\Developpement\Programmes\Privileges\Re
lease>
```

4.5. Le code du programme

Le code du programme de démonstration est le suivant :

Code du programme de démonstration

Voir le code en ligne : Lien108

5. Conclusion

Ce rapide tour d'horizon concernant les privilèges dans l'environnement Microsoft Windows est maintenant terminé, à vous de jouer et n'oubliez pas qu'en termes de sécurité, l'abus de privilèges est dangereux.

Retrouvez l'article de ram-0000 en ligne : Lien109

Sécurité informatique - Ethical Hacking - Apprendre l'attaque pour mieux se défendre

Ce livre sur la sécurité informatique (et le ethical hacking) s'adresse à tout informaticien sensibilisé au concept de la sécurité informatique mais novice en la matière. Il a pour objectif d'initier le lecteur aux techniques des attaquants pour lui apprendre comment se défendre.
Après une définition précise des différents types de hackers et de leurs objectifs, les auteurs présentent la méthodologie d'une attaque et les moyens de repérer les failles par lesquelles on peut s'insérer dans un système.
Le chapitre sur le Social Engineering, ou manipulation sociale, illustre pourquoi les failles humaines représentent plus de 60 % des attaques réussies. Les failles physiques, qui permettent un accès direct aux ordinateurs visés ainsi que les failles réseaux et Wi-Fi, sont présentées et illustrées avec à chaque fois des propositions de contre-mesures. La sécurité sur le Web est également présentée et les failles courantes identifiées à l'aide d'outils qui peuvent facilement être mis en place par le lecteur sur ses propres systèmes. L'objectif est toujours d'identifier les failles possibles pour ensuite mettre en place la stratégie de protection adaptée. Enfin, les failles systèmes sous Windows ou Linux sont recensées puis les failles applicatives avec quelques éléments pour se familiariser au langage assembleur et ainsi mieux comprendre les possibilités d'attaques.
Les auteurs de ce livre composent une équipe de personnes de conviction qui se donnent pour mission de rendre la sécurité informatique accessible à tous : "apprendre l'attaque pour mieux se défendre" est leur adage. Hackers blancs dans l'âme, ils ouvrent au lecteur les portes de la connaissance underground.

Critique du livre par Pierre Therrode

La sécurité informatique est souvent apparentée à de la magie, où le talent du magicien n'a d'égal que celui du pirate. Tout deux ont des compétences et un savoir qui en font leur réputation.
À travers cet ouvrage intitulé Sécurité Informatique -Ethical Hacking-, les sept auteurs (dont certains sont des anciens de la Hackademy) nous dévoilent leurs secrets afin de savoir comment opèrent les pirates.
Le but premier de cet ouvrage est d'apprendre les moyens d'attaques pour savoir s'en défendre et d'expliquer clairement ce qui peut être difficilement abordable à tout néophyte qui souhaite connaître ce milieu.
Au fil des chapitres, le lecteur découvrira les différentes méthodes d'élévation des privilèges. Que ce soit sur le plan humain, celui du réseau, du Web ou même du système, il sera alors question d'ingénierie sociale, de nmap, hping, aicrack pour le réseau, de Burp Suite, WebScarab pour ce qui est du Web, de JTR, ophcrack, Cain&Abel, pour le déchiffrement de mot de passe, de failles système et applicatives et avec sans oublier un tour dans le système Linux.
Enfin, les chapitres qui m'ont parus les plus intéressants sont ceux concernant les failles Web et les failles applicatives. Ces deux notions étant généralement assez difficiles à comprendre au premier abord, elles sont relativement bien expliquées.
En conclusion, un très bon livre qui met avant tout l'accent sur la technique sans pour autant déboussoler le lecteur néophyte.

Retrouvez cette critique de livre sur la page livres Sécurité : Lien110

Moteur de lumières dynamiques 2D

Cet article va vous permettre de comprendre le fonctionnement du moteur de lumières dynamiques en dimension 2 que j'ai conçu pour mon projet Holyspirit (Lien111).

1. Introduction

Notre objectif est de réaliser un moteur de lumières dynamiques en dimension 2.
Il nous faudra donc d'abord trouver un moyen de représenter une lumière et son impact sur l'environnement, et trouver ensuite un moyen de l'empêcher de traverser les murs.

Nous allons le concevoir dans un premier temps pour un monde purement en dimension 2, je vous expliquerai ensuite comment, moi, je l'ai adapté pour un jeu en 2D dimétrique (souvent appelé isométrique), donnant une impression de profondeur/hauteur.

Je précise qu'il n'est pas forcément optimisé et toujours à 100% efficace, n'hésitez pas à me faire part de vos remarques.
Mon moteur est conçu avec la SFML (Lien112), mais il est aisément adaptable à OpenGL.

Pour comprendre l'intégralité de cet article, il faut avoir des petites notions de mathématiques. Mais rassurez-vous, rien de bien méchant, il vous suffit juste de comprendre l'équation d'une droite de type y = ax + b et de savoir résoudre des simples systèmes d'équations à deux inconnues. Il faut aussi savoir ce qu'est le sinus et le cosinus d'un angle.

2. Représentation d'une lumière

Une lumière est définie par une position (abscisse et ordonnée), un rayon (nous ne ferons dans un premier temps que des lumières de type *omnidirectionnelles*, nous verrons plus tard pour faire des lumières directionnelles) et une couleur.

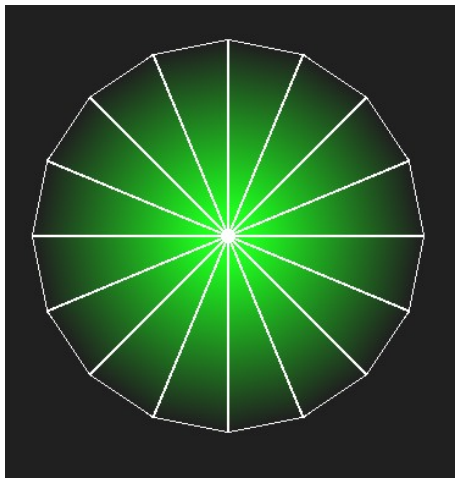
Une lampe omnidirectionnelle est une source de lumière qui projette de la lumière dans toutes les directions à la fois.

Une lumière est donc un point qui projette une couleur dans toutes les directions, sous forme de cercle, avec un dégradé jusqu'au noir en dehors du rayon.

La meilleure solution que j'ai trouvée est de découper cette lumière en triangles. Chacun de ces triangles possède un sommet en commun : le centre de la lumière.
Gérer des triangles est très simple et est une fonctionnalité de base d'OpenGL.

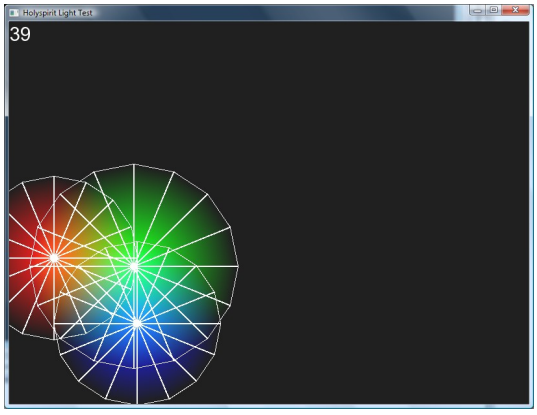
Nous pouvons donc encore ajouter une option aux lumières : leur "qualité", c'est-à-dire le nombre de triangles qui les composent à la base.

Voici un exemple de lumière de couleur verte, dont les triangles sont bordés juste pour que vous puissiez visionner le fonctionnement du moteur :



Pour représenter l'impact de ces lumières, j'ai décidé de partir sur un système d'écran virtuel qui servira de tampon. Cet écran permettra de générer une image qui pourra modifier les pixels à l'écran, pour donner une impression d'éclairage.

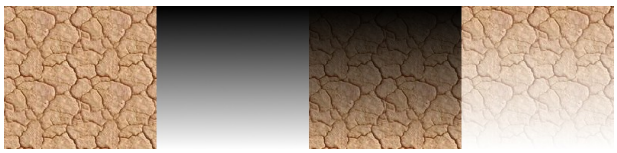
Nous allons donc afficher ces triangles en mode de rendu *additif*, c'est-à-dire que la couleur des pixels est ajoutée à la couleur des pixels déjà rendu à l'écran. Je dessine tout cela sur un fond de la couleur de la lumière ambiante voulue. Ainsi, les lumières s'additionnent entre elles, permettant des fondus.



Ce rendu final est ensuite copié en mémoire pour être finalement redessiné une fois les rendus du jeu faits, en mode de rendu multiplicatif. Cela signifie que la couleur des pixels de l'écran est multipliée par la couleur des pixels de la texture rendue. En prenant en compte, bien

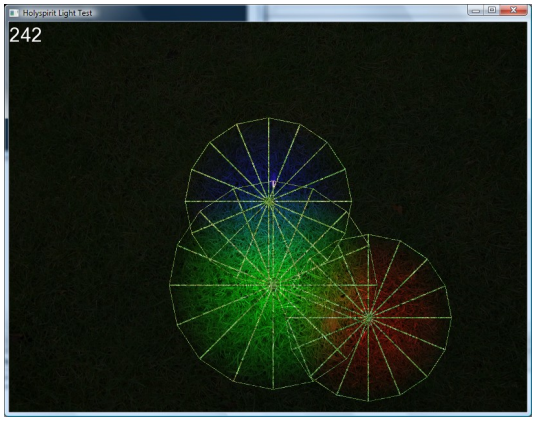
évidemment, que la couleur des pixels est définie par une valeur de 0 à 1 pour le rouge, le vert et le bleu (et non pas de 0 à 255).

Voici une image d'exemple pour bien comprendre la différence entre les différents modes de rendu :



Tout à gauche, nous avons notre image de fond, ensuite vient en deuxième lieu l'image que nous allons dessiner.
En troisième position, notre image est rendue en mode multiplicatif, la couleur des pixels de notre image de fond est multipliée par la couleur des pixels de notre image rendue.
En quatrième position, nous avons notre image qui est rendue en mode additif, la couleur de ses pixels est ajoutée aux pixels déjà existants.

Tout cela pour permettre d'assombrir tout le rendu, sauf où il y a des lumières !
Et il suffit de changer la couleur de fond pour avoir des ambiances bien sympas. Prenons par exemple une couleur orangée pour donner un effet de lever de soleil, ou un bleu foncé pour la nuit.



Ne faites pas attention aux lignes plus claires, ce sont toujours les contours de nos triangles.

Mes lumières sont gérées via un gestionnaire de lumières que j'ai conçu. Il permet d'en ajouter, d'en supprimer, de les modifier, de les déplacer, etc. C'est aussi lui qui s'occupe de faire les rendus.
Je vous conseille de coder le vôtre afin qu'il soit adapté aux technologies que vous utilisez, mais sinon, vous pouvez avoir le mien grâce au code d'exemple donné dans la partie Liens et conclusion.

Par exemple, ajouter une lumière donne ceci :

```
Light_Entity light;
light = Manager-
>Add_Dynamic_Light(sf::Vector2f(600,200),255,160,
16,sf::Color(0,255,0));
```

Les paramètres de la méthode sont la position, l'intensité, le rayon, la qualité et la couleur.

Remarquez que je précise *Dynamic*, en effet, mon gestionnaire gère les lumières dynamiques et statiques.
La seule différence vient du fait que les lumières dynamiques sont générées à chaque tour de boucle, tandis que les lumières statiques ne le sont que lors de leur création, ceci afin d'éviter de consommer des performances pour des lumières qui ne sont jamais modifiées.
Par exemple, un joueur qui lance une boule de feu va créer une lumière dynamique, en effet, celle-ci se déplace.
Tandis qu'une torche sur un mur ne bouge pas, sa lumière peut donc être statique et donc son rendu calculé uniquement au chargement de la carte.

Je peux accéder à mes lampes en faisant par exemple :

```
Manager->SetRadius(light,128 );
```

Le moteur de lumières possède une méthode Generate() qui permet de générer toutes les lumières et de les rendre à l'écran. Elle va donc boucler sur l'ensemble des lumières contenues dans le moteur et faire appel à leur propre méthode Generate(). Le code de génération d'une lumière :

```
void Light::Generate()
{
    m_shape.clear();

    float buf=(M_PI*2)/(float)m_quality;

    for(int i=0;i &lt; m_quality;++i)
    {
        AddTriangle(sf::Vector2f((float)
((float)m_radius*cos((float)i*buf))
                                 ,(float)
((float)m_radius*sin((float)i*buf))) ,
                    sf::Vector2f((float)
((float)m_radius*cos((float)(i+1)*buf))
                                 ,(float)
((float)m_radius*sin((float)(i+1)*buf))));
    }
}
```

m_shape est un tableau dynamique de polygones. C'est une fonction par défaut de la SFML, vous pouvez facilement créer une classe ou une structure utilisant des GLTriangles pour faire cela.

En gros, on divise 2pi (la circonférence d'un cercle de rayon 1) par la qualité, ainsi on obtient l'angle entre les deux côtés du triangle adjacents au centre de la lumière. Ceci nous permet de projeter ensuite les deux points extrémités de chaque triangle qui compose la lampe, grâce à la trigonométrie.
Ces points sont passés en paramètres à la fonction AddTriangle() qui va s'occuper d'ajouter le triangle à m_shape.

```
void Light::AddTriangle(sf::Vector2f
pt1,sf::Vector2f pt2)
{
    float intensity;

    m_shape.push_back(sf::Shape ());

    m_shape.back().AddPoint(0, 0,
sf::Color((int)(m_intensity*m_color.r/255),
```

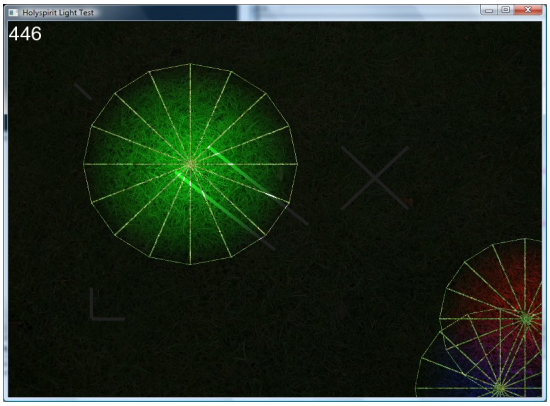
```
                                                    (int
)(m_intensity*m_color.g/255),
                                                    (int
)
(m_intensity*m_color.b/255)),sf::Color(255,255,25
5));

    intensity = m_intensity-sqrt(pt1.x*pt1.x +
pt1.y*pt1.y)*m_intensity/m_radius;
    m_shape.back().AddPoint(pt1.x, pt1.y,
sf::Color((int)(intensity*m_color.r/255),

(int)(intensity*m_color.g/255),

(int)
(intensity*m_color.b/255)),sf::Color(255,255,255)
);

    intensity = m_intensity-sqrt(pt2.x*pt2.x +
pt2.y*pt2.y)*m_intensity/m_radius;
    m_shape.back().AddPoint(pt2.x, pt2.y,
sf::Color((int)(intensity*m_color.r/255),

(int)(intensity*m_color.g/255),

(int)
(intensity*m_color.b/255)),sf::Color(255,255,255)
);

    m_shape.back().SetBlendMode(sf::Blend::Add);
    m_shape.back().SetPosition(m_position);
}
```

Je ne sais pas si OpenGL gère par défaut les modes de rendus (Add et Multiply), mais je pense que oui, étant donné que la SFML le fait nativement, via SetBlendMode().

J'ajoute les trois points qui composent le triangle à mon m_shape, avec leur couleur respective.
Je calcule déjà leur intensité en fonction de leurs distances avec le centre, vous verrez pourquoi je fais ça dans la suite du tutoriel.

3. Gestion des murs

Maintenant, nous allons voir comment gérer des murs. Ces murs sont des segments de droites que la lumière ne peut pas traverser.
Par souci de lisibilité, je vais représenter graphiquement ces murs par des lignes blanches (devenues grises à cause de la lumière ambiante).
Voici les murs que nous allons utiliser pour nos tests :

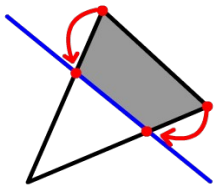


Mon gestionnaire de lumières s'occupe aussi de gérer mes murs.

Un mur est juste composé de ses deux points extrémités. La liste des murs est envoyée à la méthode AddTriangle() de nos lumières.
Il nous suffit maintenant de calculer les intersections entre les murs et les triangles qui composent la lumière, afin de "couper" celle-ci et l'empêcher de traverser ceux-ci.

Il existe trois façons dont le mur peut couper un triangle (si l'on ne prend pas en compte des exceptions où des points se confondent).

Premièrement, le mur peut couper de part en part le triangle, comme ceci :



Les lignes noires représentent le triangle, la ligne bleue le mur et les points rouges les intersections.
La partie grisée est la partie du triangle qui disparaît.

Cette situation est la plus simple, il nous suffit de déplacer les deux sommets extrémités du triangle afin de les ramener au niveau des intersections entre les côtés adjacents et le mur.

Je nommerai à l'avenir ces deux côtés qui partent du centre *côtés adjacents*.
Le côté du triangle le plus éloigné du centre sera nommé *côté extrémité*.

Pour trouver ces deux points, je calcule les équations cartésiennes des quatre droites sur lesquelles sont alignés nos trois côtés du triangle et notre mur. Je calcule l'intersection entre le mur et chacun des trois côtés grâce à cette fonction :

```
sf::Vector2f Intersect(sf::Vector2f p1,
sf::Vector2f p2, sf::Vector2f q1, sf::Vector2f
q2)
{
    sf::Vector2f i;

    float a = (p2.y - p1.y) / (p2.x - p1.x);
    float b = p1.y - p1.x * a;

    float c = (q2.y - q1.y) / (q2.x - q1.x);
    float d = q1.y - q1.x * c;

    i.x = (d-b)/(a-c);
    i.y = a * i.x + b;

    return i;
}
```

Mes deux droites sont représentées sous forme d'équations y = ax + b et y = cx + d.

Je calcule a et c, le coefficient de la pente, en prenant le rapport entre la hauteur et la largeur qui séparent les deux points.
Je calcule ensuite b et d en manipulant l'équation :
$y = ax + b \Rightarrow b = y - ax$

Je calcule mon point d'intersection grâce à ce tout petit système d'équations :
$y = ax + b$
$y = cx + d$

$ax + b = cx + d$
$ax - cx = d - b$
$x = (d - b)/(a - c)$

$y = ax + b$ (x étant calculé juste au-dessus)

Et voilà, nous avons les coordonnées en abscisse et ordonnée de notre point d'intersection !

Ce code n'est pas "sécurisé", il faut maintenant rajouter la gestion des erreurs.
Les habitués auront tout de suite remarqué les divisions par 0.

```
sf::Vector2f Intersect(sf::Vector2f p1,
sf::Vector2f p2, sf::Vector2f q1, sf::Vector2f
q2)
{
    sf::Vector2f i;

    if((p2.x - p1.x) == 0 && (q2.x - q1.x) == 0)
        i.x = 0, i.y = 0;
    else if((p2.x - p1.x) == 0)
    {
        i.x = p1.x;

        float c = (q2.y - q1.y) / (q2.x - q1.x);
        float d = q1.y - q1.x * c;

        i.y = c * i.x + d;
    }
    else if((q2.x - q1.x) == 0)
    {
        i.x = q1.x;

        float a = (p2.y - p1.y) / (p2.x - p1.x);
        float b = p1.y - p1.x * a;

        i.y = a * i.x + b;
    }
    else
    {
        float a = (p2.y - p1.y) / (p2.x - p1.x);
        float b = p1.y - p1.x * a;

        float c = (q2.y - q1.y) / (q2.x - q1.x);
        float d = q1.y - q1.x * c;

        i.x = (d-b)/(a-c);
        i.y = a * i.x + b;
    }

    return i;
}
```

Nous calculerons toujours ces trois points d'intersection.

C'est en vérifiant s'ils appartiennent aux segments de droites que nous saurons dans quel cas de coupe nous sommes.
Ici, avec le mur qui coupe le triangle de part en part, les deux points d'intersection des côtés adjacents appartiendront aux deux segments.

Si le point d'intersection n'appartient pas au mur, c'est que celui-ci ne coupe pas ce triangle de la lumière.

Pour vérifier cela, j'utilise ce code :

```
sf::Vector2f Collision(sf::Vector2f p1,
sf::Vector2f p2, sf::Vector2f q1, sf::Vector2f
q2)
{
    sf::Vector2f i;
    i = Intersect(p1, p2, q1, q2);

    if(((i.x >= p1.x - 0.1 && i.x <= p2.x + 0.1)
     || (i.x >= p2.x - 0.1 && i.x <= p1.x + 0.1))
    && ((i.x >= q1.x - 0.1 && i.x <= q2.x + 0.1)
     || (i.x >= q2.x - 0.1 && i.x <= q1.x + 0.1))
    && ((i.y >= p1.y - 0.1 && i.y <= p2.y + 0.1)
     || (i.y >= p2.y - 0.1 && i.y <= p1.y + 0.1))
    && ((i.y >= q1.y - 0.1 && i.y <= q2.y + 0.1)
     || (i.y >= q2.y - 0.1 && i.y <= q1.y +
0.1)))
        return i;
    else
        return sf::Vector2f (0,0);
}
```

Dans un cas parfait, il nous faudrait juste vérifier en x, hélas il y a parfois des décimales qui se perdent, je prends donc aussi une marge d'erreur de 0.1.

Cette fonction retourne 0,0 si il n'y a pas de point d'intersection entre les deux segments de droites.

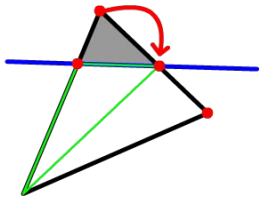
Notre méthode AddTriangle() donne donc ceci :

```
void Light::AddTriangle(sf::Vector2f
pt1,sf::Vector2f pt2,std::vector <Wall> &m_wall)
{
    for(std::vector<Wall>::iterator
IterWall=m_wall.begin()+minimum_wall ; IterWall!
=m_wall.end() ; ++IterWall)
    {
        // l1 et l2 sont les positions relatives
au centre de la lumière des deux extrémités du
mur
        sf::Vector2f l1(IterWall->pt1.x-
m_position.x, IterWall->pt1.y-m_position.y);
        sf::Vector2f l2(IterWall->pt2.x-
m_position.x, IterWall->pt2.y-m_position.y);

        //Point d'intersection entre le mur et le
côté qui part du centre jusqu'à l'extrémité 1,
c'est donc un des deux côtés adjacents.
        sf::Vector2f m = Collision(l1, l2,
sf::Vector2f(0,0), pt1);
        //Idem que juste au dessus, mais avec
l'autre extrémité.
        sf::Vector2f n = Collision(l1, l2,
sf::Vector2f(0,0), pt2);
        //Segment qui relie les deux extrémités,
c'est le côté extrémité.
        sf::Vector2f o = Collision(l1, l2, pt1,
pt2);
```

```
        //Vérification que les deux points
d'intersections appartiennent bien aux deux côtés
adjacents du triangle.
        if((m.x != 0 || m.y != 0) && (n.x != 0 ||
n.y != 0))
            pt1 = m, pt2 = n;
    }

    // METHODE DE RENDU LA MÊME QUE PLUS HAUT
    ...
    // FIN DE LA METHODE DE RENDU
}
```

Dans un jeu complexe, il est conseillé de partitionner l'espace (ou le plutôt le plan, dans notre cas) afin d'éviter de calculer les intersections murs/lumières si les deux objets sont trop éloignés.

Maintenant, attaquons-nous au deuxième cas de figure :



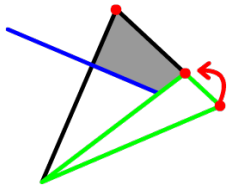
Le mur coupe un des côtés adjacents et le côté extrémité.

Dans ce cas, il nous faut découper le triangle en deux, donc modifier le triangle en cours et en ajouter un autre (en vert).
La méthode AddTriangle() devient donc une méthode inclusive.

D'un point de vue programmation, il suffit de vérifier que l'un de nos deux points d'intersection n'appartient pas à un côté adjacent. Sinon, on regarde s'il y a une intersection entre le côté extrémité et le mur.
On ajoute alors un triangle et on déplace l'extrémité du triangle actuel pour la ramener à l'intersection entre le mur et le côté extrémité. N'hésitez pas à regarder à nouveau le schéma pour comprendre.

```
if((m.x != 0 || m.y != 0) && (n.x != 0 || n.y !=
0))
    pt1 = m, pt2 = n;
else
{
    if((m.x != 0 || m.y != 0) && (o.x != 0 || o.y
!= 0))
        AddTriangle(m ,o , w, m_wall), pt1 = o;

    if((n.x != 0 || n.y != 0) && (o.x != 0 || o.y
!= 0))
        AddTriangle(o ,n , w, m_wall), pt2 = o;
}
```

Enfin, troisième et dernière possibilité :



L'extrémité du mur arrive dans le triangle.

Si cela arrive, nous divisons le triangle en deux.
Pour vérifier cela, il suffit de regarder d'abord si l'une des deux extrémités est plus proche du centre que la taille du rayon.
Et de regarder ensuite si le point d'intersection entre la droite projetée depuis le centre jusqu'à l'extrémité du mur et le côté extrémité du triangle se trouve entre les deux côtés adjacents du triangle.

On coupe le triangle au niveau du point d'intersection entre la droite projetée et le côté extrémité.
Nous nous retrouvons ensuite dans le premier cas pour l'un des deux triangles qui sera alors coupé de part en part par le mur.

Il faut donc impérativement tester ce premier cas avant les autres.

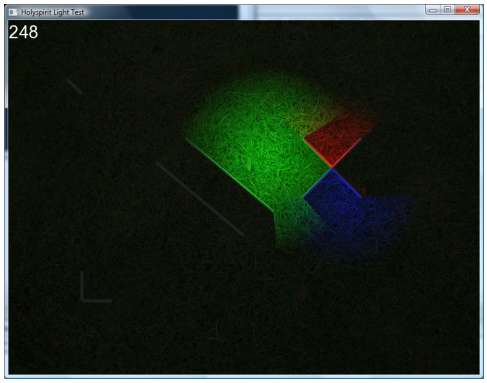
```
void Light::AddTriangle(sf::Vector2f
pt1,sf::Vector2f pt2, int
minimum_wall,std::vector <Wall> &m_wall)
{
    int w = minimum_wall;

    for(std::vector<Wall>::iterator
IterWall=m_wall.begin()+minimum_wall;IterWall!
=m_wall.end();++IterWall,++w)
    {
        // l1 et l2 sont les positions relatives
au centre de la lumière des deux extrémités du
mur
        sf::Vector2f l1(IterWall->pt1.x-
m_position.x, IterWall->pt1.y-m_position.y);
        sf::Vector2f l2(IterWall->pt2.x-
m_position.x, IterWall->pt2.y-m_position.y);

        if(l1.x * l1.x + l1.y * l1.y < m_radius *
m_radius)
        {
            sf::Vector2f i =
Intersect(pt1,pt2,sf::Vector2f (0,0),l1);

            if((pt1.x > i.x && pt2.x < i.x) ||
(pt1.x < i.x && pt2.x > i.x))
            if((pt1.y > i.y && pt2.y < i.y) ||
(pt1.y < i.y && pt2.y > i.y))
                if(l1.y > 0 && i.y > 0 || l1.y <
0 && i.y < 0)
                if(l1.x > 0 && i.x > 0 || l1.x <
0 && i.x < 0)
                AddTriangle(i, pt2, w, m_wall),
pt2 = i;
        }
        if(l2.x * l2.x + l2.y * l2.y < m_radius *
m_radius)
        {
```

```
            sf::Vector2f i =
Intersect(pt1,pt2,sf::Vector2f (0,0),l2);

            if((pt1.x > i.x && pt2.x < i.x) ||
(pt1.x < i.x && pt2.x > i.x))
            if((pt1.y > i.y && pt2.y < i.y) ||
(pt1.y < i.y && pt2.y > i.y))
                if(l2.y > 0 && i.y > 0 || l2.y <
0 && i.y < 0)
                if(l2.x > 0 && i.x > 0 || l2.x <
0 && i.x < 0)
                AddTriangle(pt1, i, w, m_wall),
pt1 = i;
        }

        sf::Vector2f m = Collision(l1, l2,
sf::Vector2f(0,0), pt1);
        sf::Vector2f n = Collision(l1, l2,
sf::Vector2f(0,0), pt2);
        sf::Vector2f o = Collision(l1, l2, pt1,
pt2);

        if((m.x != 0 || m.y != 0) && (n.x != 0 ||
n.y != 0))
            pt1 = m, pt2 = n;
        else
        {
            if((m.x != 0 || m.y != 0) && (o.x !=
0 || o.y != 0))
                AddTriangle(m ,o , w, m_wall),
pt1 = o;

            if((n.x != 0 || n.y != 0) && (o.x !=
0 || o.y != 0))
                AddTriangle(o ,n , w, m_wall),
pt2 = o;
        }
    }

    float intensity;

    m_shape.push_back(sf::Shape ());

    m_shape.back().AddPoint(0, 0,
sf::Color((int)(m_intensity*m_color.r/255),
                                            (int
)(m_intensity*m_color.g/255),
                                            (int
)
(m_intensity*m_color.b/255)),sf::Color(255,255,25
5));

    intensity=m_intensity-sqrt(pt1.x*pt1.x +
pt1.y*pt1.y)*m_intensity/m_radius;
    m_shape.back().AddPoint(pt1.x, pt1.y,
sf::Color((int)(intensity*m_color.r/255),

(int)(intensity*m_color.g/255),

(int)
(intensity*m_color.b/255)),sf::Color(255,255,255)
);

    intensity=m_intensity-sqrt(pt2.x*pt2.x +
pt2.y*pt2.y)*m_intensity/m_radius;
    m_shape.back().AddPoint(pt2.x, pt2.y,
sf::Color((int)(intensity*m_color.r/255),

(int)(intensity*m_color.g/255),

(int)
(intensity*m_color.b/255)),sf::Color(255,255,255)
);
```

```
    m_shape.back().SetBlendMode(sf::Blend::Add);
    m_shape.back().SetPosition(m_position);
}
```

Remarquez l'ajout d'un paramètre minimum_wall, celui-ci permet de revenir où l'on était arrivé dans la liste des murs lors de l'ajout d'un triangle supplémentaire. En effet, on peut considérer que le nouveau triangle a déjà été vérifié avec les mêmes murs que l'ancien triangle qui le contenait.

Et voici ce que cela donne au final :

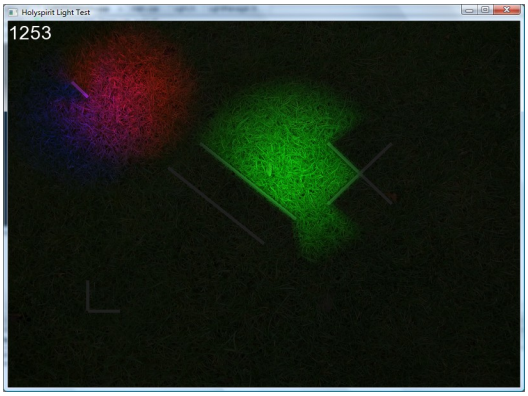


C'est-y pas magnifique ?

Pour ma part, j'ajoute un filtre de flou sur le rendu des lumières, je trouve cela plus esthétique.

Ce filtre flou est en fait un shader appliqué lors du rendu final en mode multiplicatif.
Voici le code de ce shader :

```
uniform sampler2D texture;
uniform float offset;

void main()
{
        vec2 offx = vec2(offset, 0.0);
        vec2 offy = vec2(0.0, offset);

        vec4 pixel = texture2D(texture,
gl_TexCoord[0].xy)               * 1 +
                     texture2D(texture,
gl_TexCoord[0].xy - offx)        * 2 +
                     texture2D(texture,
gl_TexCoord[0].xy + offx)        * 2 +
                     texture2D(texture,
gl_TexCoord[0].xy - offy)        * 2 +
                     texture2D(texture,
gl_TexCoord[0].xy + offy)        * 2 +
                     texture2D(texture,
gl_TexCoord[0].xy - offx - offy) * 1 +
                     texture2D(texture,
gl_TexCoord[0].xy - offx + offy) * 1 +
                     texture2D(texture,
gl_TexCoord[0].xy + offx - offy) * 1 +
                     texture2D(texture,
gl_TexCoord[0].xy + offx + offy) * 1;

        gl_FragColor =  gl_Color * (pixel /
13.0);
}
```



4. Les lumières directionnelles

Les lumières directionnelles, au contraire des lumières de type omnidirectionnel, ne projettent de la lumière que dans une seule direction.
Elles seront donc composées par un seul triangle.

Nos *Directional_light* héritent de *Light*. On a juste besoin de leur rajouter deux paramètres : leur angle et leur angle d'ouverture.
Nous devons aussi surcharger la méthode Generate(), comme ceci, afin de ne plus générer qu'un seul triangle :

```
void
Directional_light::Generate(std::vector<Wall>
&m_wall)
{
    m_shape.clear();

    float angle     = m_angle * M_PI / 180;
    float o_angle   = m_opening_angle * M_PI /
180;

    AddTriangle(sf::Vector2f((m_radius*cos(angle
+ o_angle * 0.5))
                          ,(m_radius*sin(angle
+ o_angle * 0.5))) ,
                sf::Vector2f((m_radius*cos(angle
- o_angle * 0.5))
                          ,(m_radius*sin(angle
- o_angle * 0.5))),0,m_wall);
}
```

Mes angles sont en degrés, je dois donc les convertir en gradients pour pouvoir les utiliser avec les fonctions sinus et cosinus.

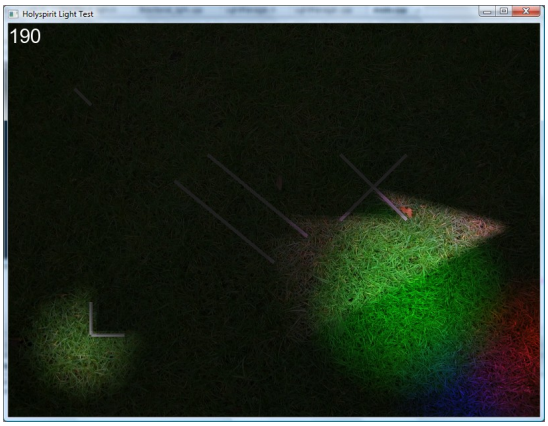
J'ai dû aussi faire pas mal de modifications dans mon gestionnaire de lumières, comme passer les Light en pointeurs afin de profiter du polymorphisme dans mes tableaux dynamiques.

Voici un exemple pour ajouter une lumière directionnelle :

```
directional_light = Manager-
>Add_Dynamic_Directional_Light(sf::Vector2f(750,3
10),255,384,180,45,sf::Color(255,128,255));
```

Les paramètres sont la position du projecteur, l'intensité, la taille du triangle, l'angle, l'angle d'ouverture et la couleur.

Ça ne demande pas vraiment plus de travail que ça d'un point de vue conception, le tout étant de créer le gestionnaire.



5. Annexe : intégration à Holyspirit

Maintenant que notre moteur de lumières est créé, il faut l'implanter dans le jeu.
Je ne vais pas expliquer ici comment je gère ça dans mon code, ni comment c'est intégré par rapport aux ressources, mais je vais surtout expliquer comment l'utiliser pour le rendre compatible avec une méthode de rendu en 2D dimétrique (souvent appelé 2D isométrique car ça y ressemble fort).

La première grosse différence vient de la profondeur. En effet, nous n'avons plus une vue du sol d'en haut, mais légèrement de côté. Étant donné que nous sommes en dimétrique, nous savons que l'effet de profondeur est créé par des tiles écrasés de manière à avoir deux fois leur hauteur dans leur longueur.
Il nous suffit donc "d'écraser" aussi nos lumières, en divisant leurs coordonnées de rendu par deux.
Nous obtenons ceci :

La deuxième différence vient de la gestion de la hauteur des murs.
Je dois donc ajouter une hauteur à mes murs et projeter la lumière dessus.

La première étape consiste à vérifier que la lumière est devant le mur.

```
bool devant = false;
if ( 0>=(pt1.y) - (pt1.x)*(((pt1.y)-(pt2.y))/
((pt1.x)-(pt2.x))))
   devant = true;
```

Il me suffit de vérifier si l'ordonnée à l'origine (b) est plus petite que 0 dans l'équation de ma droite y = ax + b.
Soit b = y – ax
y étant la coordonnée en y d'un de mes points, x la coordonnée en x d'un de mes points et je retrouve a en prenant le rapport de la différence en y sur la différence en x de mes deux points.
L'ordonnée à l'origine doit être négative car l'axe des y va vers le bas dans la SFML.

Maintenant que nous savons si le mur est devant, il nous suffit de rajouter un parallélépipède à m_shape.

```
if (devant)
  if (intensity>1||intensity2>1)
  {
    m_shape.push_back(sf::Shape ());

    m_shape.back().AddPoint(pt1.x,pt1.y/2,
sf::Color((int)(intensity*m_color.r/255),(int)
(intensity*m_color.g/255),(int)
(intensity*m_color.b/255)));
    m_shape.back().AddPoint(pt1.x,pt1.y/2-96 *
sin(intensity /m_intensity*M_PI_2),
sf::Color(0,0,0));
    m_shape.back().AddPoint(pt2.x,pt2.y/2-96 *
sin(intensity2/m_intensity*M_PI_2),
sf::Color(0,0,0));
    m_shape.back().AddPoint(pt2.x,pt2.y/2,
sf::Color((int)(intensity2*m_color.r/255),(int)
(intensity2*m_color.g/255),(int)
(intensity2*m_color.b/255)));

    m_shape.back().SetBlendMode(sf::Blend::Add);

    m_shape.back().SetPosition(m_position.x,m_pos
ition.y/2);
  }
```

Remarquez que je projette la hauteur avec un sinus, cela me permet de donner une forme de demi-ellipse à ma projection.

J'applique ensuite mon rendu par-dessus tout le jeu. Cela donne parfois des incohérences, comme juste le dessus d'un personnage qui est éclairé au lieu de tout son corps, mais cela est plus performant que de calculer deux rendus (un pour le sol et un pour les entités et murs).

Et voici le résultat final :

Si vous voulez le voir en mouvement, je vous invite à regarder cette vidéo : Lien113

6. Liens et conclusion

J'espère que cet article vous aura été utile et, au risque de me répéter, n'hésitez pas à me faire part de vos remarques ! Si un jour vous concevez un jeu ou autre avec mon moteur, ou en vous inspirant de celui-ci, n'hésitez pas à m'envoyer un mail, ça me fera plaisir de voir que j'ai pu être utile.

Et maintenant, voici quelques liens utiles :

- voir l'article original sur mon blog : Lien114 ;
- télécharger le programme d'exemple : Lien115 ;
- télécharger Holyspirit (pour ceux qui veulent tester pour voir ce que cela donne dans un jeu.) : Lien116.

Retrouvez l'article de Naisse Grégoire en ligne : Lien117

Les extensions OpenGL

Le but de l'article est d'expliquer de façon détaillée ce qu'est une extension OpenGL ainsi que son utilisation.

1. Introduction

1.1. Pourquoi des extensions ?

Les extensions dans OpenGL permettent de rajouter des fonctionnalités sans modifier le cœur d'OpenGL (norme donnée par Khronos). Les extensions sont produites par les entreprises enregistrées dans le groupe Khronos telles que Nvidia, ATI, Apple et bien d'autres. Bien souvent, une fois que l'extension est reconnue comme essentielle, elle est intégrée dans le cœur d'OpenGL et la version d'OpenGL change. La liste des extensions pour OpenGL peut être trouvée en bas de cette page : Lien118. Il faut savoir qu'aucun constructeur n'est tenu d'inclure des extensions dans ses cartes graphiques. En effet pour être conforme à la norme OpenGL, il suffit juste de correctement implémenter le cœur d'OpenGL. Cela signifie que rien ne vous certifie que vous aurez une extension dans la machine exécutant votre programme.

1.2. Pourquoi un tutoriel ?

Principalement pour expliquer précisément ce que sont les extensions et pour savoir les utiliser dans tous les cas, avec ou sans bibliothèques externes.

1.3. Prérequis

Il vous sera utile de savoir comment créer un projet OpenGL ainsi que d'ouvrir une fenêtre OpenGL. Vous pouvez trouver des tutoriels sur ce point en suivant les liens suivants :

- WinAPI : Lien119 ;
- SDL : Lien120 ;
- Glut : Lien121.

1.4. Noms des extensions

Comme je l'ai dit un peu avant, différentes sociétés développent leurs extensions. Il existe une norme de nommage pour les extensions qui permet de connaître la bibliothèque intégrant l'extension, le nom de la compagnie à l'origine de l'extension et, finalement, le nom de l'extension.

Le nom de l'extension est constitué comme suit :

- GL _ ID-DEVELOPPEUR_nom-de-l'extension ;
- GLX _ ID-DEVELOPPEUR_nom-de-l'extension ;
- WGL _ ID-DEVELOPPEUR_nom-de-l'extension.

GL / GLX et WGL désignent la bibliothèque sur laquelle l'extension agit. Nous ne nous intéressons qu'aux extensions OpenGL (donc GL) mais le principe est exactement le même pour les autres.

Rappel : GLX, WGL mais aussi AGL et PGL sont des bibliothèques qui ont pour rôle de faire le lien entre OpenGL et le système de fenêtre du système que vous utilisez : 'X' pour le serveur X présent entre autres sous Linux, 'W' pour Windows, 'A' pour Apple et 'P' pour IBM OS/2 Warp.

ID-DEVELOPPEUR est le nom de la compagnie qui a créé la norme de l'extension. On peut entre autres voir les noms suivants :

- ARB pour Architecture Review Board (qui correspond au Comité d'évaluation de l'architecture d'OpenGL. C'est le groupe qui valide les spécifications d'OpenGL) ;
- EXT pour les extensions mises au point par un groupe de compagnies, sans particularité quelconque ;
- NV pour Nvidia ;
- ATI pour ATI, maintenant appelé AMD ;
- SGI pour Silicon Graphic International (la société à l'origine d'OpenGL).

Et j'en passe.

Et à la fin, le nom de l'extension, qui se doit d'être clair.

Voici quelques exemples tirés de cette page (Lien122) :

- GL_ARB_half_float_pixel : Lien123 ;
- GL_ARB_vertex_shader : Lien124 ;
- GL_NV_texture_shader2 : Lien125 ;
- GL_EXT_vertex_array_bgra : Lien126.

1.5. La vie des extensions

Comme il a été dit, les constructeurs présentent leurs extensions. Celles-ci ne sont pas réservées à la compagnie à l'origine de l'extension et peuvent être implémentées par toute autre compagnie. Une fois qu'une extension est présentée par une compagnie (ou un ensemble de compagnies) elle va être revue par l'ARB. Lorsque l'extension sera validée par l'ARB elle changera de nom pour prendre les lettres ARB, à la place des lettres désignant le constructeur. Il y a de très fortes chances que les extensions ayant obtenu cette ratification soient implémentées sur toutes les cartes sorties après le jour de cette ratification. Une fois qu'une extension a atteint ce stade, il est probable de voir cette extension dans la prochaine version d'OpenGL en intégrant le cœur d'OpenGL (et qu'elle soit donc incluse d'office sur toutes les cartes compatibles avec cette version d'OpenGL).

2. Utiliser une extension et ses fonctions associées

2.1. Récupérer la liste des extensions

Comme il est dit en introduction, chaque machine embarque son lot d'extensions, et une extension présente dans une machine, peut ne pas l'être dans une autre. C'est pourquoi il est important de savoir récupérer la liste des extensions présentes dans la machine, sinon votre programme va vouloir exécuter des morceaux de code qu'il n'y a pas (ça va crasher !).

Pour ce faire, nous utilisons la commande glGetString() avec comme paramètre 'GL_EXTENSIONS' : Lien127.

Bien que cette fonction puisse être très simple, il faut tout de même que votre fenêtre OpenGL soit initialisée pour que cela fonctionne, sinon la fonction provoquera une erreur (récupérable avec glGetError() (Lien128)).

Afficher la liste des extensions supportées

```
void displayExtensions(void)
{
    printf("%s\n",glGetString(GL_EXTENSIONS));
}
```

Bien sûr, on pourrait améliorer l'affichage de la liste des extensions (en remplaçant les espaces par des sauts de ligne par exemple, mais cela ne nous est pas utile pour la suite du tutoriel).

2.2. Savoir si l'extension est présente

Maintenant, il faut faire en sorte que notre programme sache si les extensions dont nous avons besoin sont présentes. Si elles ne le sont pas, le programmeur pourra soit arrêter le programme avec un message d'erreur, soit passer outre et faire en sorte de ne pas utiliser toutes les fonctionnalités du programme.

Pour cela nous allons analyser la chaîne de caractères retournée par glGetString(GL_EXTENSIONS).

Un simple test n'utilisant que strstr() ne suffit pas ! Prenons le cas des extensions 'GL_ARB_draw_buffers' et 'GL_ARB_draw_buffers_blend'. Si vous cherchez 'GL_ARB_draw_buffers' et que vous n'avez **QUE** 'GL_ARB_draw_buffers_blend', strstr() vous retournera un résultat valide bien que vous n'ayez pas l'extension.

Savoir si une extension donnée est présente sur la machine

```
char hasExtension(const char* extensionName)
{
/*
    Le cast est nécessaire à cause du type de pointeur retourné
    ( GLubyte* correspondant à un const unsigned char* )
*/
```

```
    const char* extensionsList = (const
char*)glGetString(GL_EXTENSIONS);
    char* extensionMatch = NULL;

    while ( (extensionMatch =
strstr(extensionsList, extensionName)) != NULL )
    {
        // Vérification: avons-nous la même
chaine que l'extension voulue ?
        if
( strncmp(extensionMatch,extensionName,strlen(ext
ensionName)) == 0 )
        {
            // Vérification: le nom trouvé se
termine-t-il bien ici ?
            if
( extensionMatch[strlen(extensionName)] == ' ')
            {
                return 1;
            }
        }

        // On déplace le pointeur de recherche
après la chaine trouvée pour continuer la
recherche
        extensionsList = extensionMatch +
strlen(extensionMatch);
    }

    return 0;
}
```

L'utilisation est la suivante :

```
printf("Est ce que j'ai les Geometry Shader 4 : ?
-> ");
if ( hasExtension("GL_ARB_geometry_shader4") )
{
    printf("OUI\n");
}
else
{
    printf("NON\n");
}
```

2.3. Récupérer l'adresse d'une fonction

Lorsque nous voulons utiliser les fonctions d'une extension, la seule chose que nous connaissons c'est son prototype, écrit dans le fichier d'en-têtes des extensions d'OpenGL : Lien129.

En fait, le programme ne connaît pas encore les adresses des fonctions des extensions, car comme je vous l'ai dit, ces extensions peuvent être présentes, ou pas. De ce fait, nous allons devoir récupérer l'adresse de la fonction nous-même.

Pour ce faire, les bibliothèques qui se chargent de l'ouverture de la fenêtre OpenGL nous propose une fonction ayant ce but. Son nom est *GetProcAddress (je mets une étoile avant le nom de la fonction car il y a généralement des lettres avant GetProcAddress() selon la bibliothèque que vous utilisez).

La fonction va retourner un pointeur sur une fonction, qu'il faudra caster pour que ce soit un pointeur sur le prototype de la fonction réelle (comme indiqué dans la documentation ou dans le fichier d'en-têtes). Si vous ne faites pas ce cast, le compilateur n'acceptera pas que vous passiez des paramètres à une fonction qui pour lui, ne prend pas de paramètres. Dans le cas contraire (utilisation de la fonction sans avoir fait le cast, sans paramètres alors qu'elle est censée en avoir) un bug se produira.

Finalement, si la fonction voulue n'est pas trouvée, *GetProcAddress() retournera un pointeur NULL.

Prenons la fonction ayant le prototype suivant (pris du glext.h) :

```
GLAPI void APIENTRY glFramebufferTextureFaceARB
(GLenum target, GLenum attachment, GLuint
texture, GLint level, GLenum face);
```

avec un peu plus bas dans le fichier :

```
typedef void (APIENTRYP
PFNGLFRAMEBUFFERTEXTUREFACEARBPROC) (GLenum
target, GLenum attachment, GLuint texture, GLint
level, GLenum face);
```

Le premier prototype n'est inclus que si GL_GLEXT_PROTOTYPES est défini. Mais cela ne peut marcher que si la fonction est obligatoirement dans le code du pilote de la carte graphique. Sinon, cela ne compile pas (erreur de liaison, le code de la fonction n'est pas trouvé).

Pour que la fonction soit opérationnelle, nous définissons le pointeur comme suit :

```
PFNGLFRAMEBUFFERTEXTUREFACEARBPROC
glFramebufferTextureFaceARB_Fct;
```

Et afin de récupérer le pointeur :

```
glFramebufferTextureFaceARB_Fct =
(PFNGLFRAMEBUFFERTEXTUREFACEARBPROC)glutGetProcAd
dress("glFramebufferTextureFaceARB");
```

Il est tout de même préférable et même recommandé de vérifier si votre pointeur retourné n'est pas NULL !

Une fois que cela est fait la fonction peut-être utilisée comme toute autre :

```
glFramebufferTextureFaceARB_Fct(param1,param2,par
am3,param4,param5);
```

Si les types PFNGL* ne sont pas reconnus, c'est que vous avez un fichier d'en-têtes trop vieux. Vous pouvez le changer, ou alors, définir le type (en copiant le morceau du fichier d'en-têtes en question) dans votre propre fichier, mais n'oubliez pas de définir correctement le APIENTRY aussi (on peut le recopier à partir du fichier d'en-têtes d'OpenGL (Lien130)).

2.4. Information sur les fonctions dans le cœur d'OpenGL > 1.1

Malheureusement, Microsoft Windows n'a jamais été distribué avec une version d'OpenGL supérieure à la version 1.1. Cela veut dire que pour avoir, par exemple, les shaders (inclus dans le cœur de la version 2.0) nous allons devoir récupérer les adresses des fonctions comme si elles étaient encore des extensions.

Du coup, il est aussi important de vérifier la version d'OpenGL que vous utilisez (avec glGetString(GL_VERSION))

3. Bibliothèques externes

3.1. GLEW

GLEW est une bibliothèque qui permet une gestion simplifiée des extensions (Lien131). La bibliothèque fait elle-même tout le processus que j'ai expliqué auparavant.

Après la création de la fenêtre, GLEW doit être initialisé afin que la bibliothèque puisse assigner et définir les pointeurs sur les extensions ainsi que d'autres variables permettant de connaître facilement la version d'OpenGL présente.

Initialisation GLEW

```
GLenum err = glewInit();
if (GLEW_OK != err)
{
  /* Problème : glewInit a échoué, quelquechose va sérieusement mal */
  fprintf(stderr, "Erreur : %s\n", glewGetErrorString(err));
  ...
}
```

Il reste toujours à vérifier si les extensions voulues sont présentes :
depuis la version 1.1.0 de GLEW vous pouvez savoir si une extension est présente grâce aux variables ayant la forme GLEW_{nom_de_l_extension}.

Est-ce que l'extension GL_ARB_vertex_program est présente ?

```
if (GLEW_ARB_vertex_program)
{
  /* C'est bon, vous pouvez utilisé l'extension ARB_vertex_program. */
  glGenProgramsARB(...);
}
```

Pour plus d'informations sur les possibilités de la bibliothèque, je vous invite à voir ce lien : Lien132 (en anglais).

L'inclusion du fichier d'en-têtes de GLEW doit se faire avant les autres inclusions relatives à OpenGL ou à la bibliothèque qui crée la fenêtre. Si vous ne le faites pas, vous aurez des erreurs de compilation.

3.2. GLEE

GLEE est une bibliothèque qui a le même rôle que GLEW. À vrai dire, GLEE existait avant l'apparition de GLEW mais n'intègre pas les dernières extensions et fonctionnalités d'OpenGL (Lien133).

L'utilisation de GLEE est simple. Il suffit juste d'inclure le fichier d'en-têtes de la bibliothèque et de vérifier si l'extension est présente :

Utilisation de GLEE

```
if (GLEE_ARB_multitexture)    // Est-ce que le support des multitexture est présent
{
  glMultiTexCoord2fARB(...);  // Aucun risque d'utiliser la fonction.
}
else
{
  // Retour d'erreur
}
```

4. Lire la documentation d'une extension

Savoir lire une documentation est la chose la plus importante que doit savoir faire un programmeur.

La documentation des extensions est, d'après moi, loin d'être la plus simple. Elle est contenue dans un fichier texte presque sans aucune mise en page.

Le fichier a la structure générale suivante :

- Name : le nom de l'extension ;
- Name string : le nom comme on le verra dans la liste des extensions lors de l'exécution de notre programme ;
- Contact : les personnes à contacter au sujet de l'extension ;
- Status : l'état actuel de l'extension (approuvée ou non pas l'ARB) ;
- Version : la version actuelle de l'extension ;
- Number : numéro de l'extension ;
- Dependencies : les dépendances que nécessite l'extension et la liste des extensions impactées par celle-ci ;
- Overview : une brève description de l'extension ;
- New Procedures and Functions : les nouvelles fonctions qu'implémente l'extension ;
- New Tokens : les nouvelles variables qu'introduit l'extension ;
- Additions ; suivi de l'emplacement de l'ajout : ajout de documentation dans le document de la norme d'OpenGL. Ici se trouve le détail complet de l'extension et de son implémentation ;
- Dependencies ; et le nom de l'extension que cela affecte : explication de l'ajout des fonctionnalités de cette extension, au sein d'autres extensions ;
- Errors : liste des erreurs que peut provoquer l'extension ;
- New states : liste des nouveaux états provoqués par l'extension ;
- Issues : toutes les questions qui ont été posées durant la conception pour clarifier des détails sur l'extension ;
- Revision history : la liste des changements au fil des versions.

Il se peut qu'il y ait une légère différence avec cette énumération, mais dans la globalité, cela reste toujours de cette forme-là.

5. Liens

- site d'OpenGL : Lien134 ;
- site de Khronos : Lien135 ;
- documentation officielle sur les extensions : Lien136 ;
- liste et documentation des extensions : Lien137 ;
- fichier d'entête des extensions : Lien129 ;
- page du projet GLEW : Lien138 ;
- page du projet GLEE : Lien133 ;
- explication GLX / AGL / WGL / PGL : Lien139.

Retrouvez l'article d'Alexandre Laurent en ligne : Lien140

Liens

Lien1 : http://www.jroller.com/scolebourne/entry/the_next_big_jvm_language1
Lien2 : http://www.jroller.com/scolebourne/entry/two_features_for_bijava
Lien3 : http://www.jroller.com/scolebourne/entry/checked_exceptions_bijava
Lien4 : http://blog.developpez.com/adiguba/p8434/java/unchecked-checked-exception/
Lien5 : http://blog.developpez.com/adiguba/p9316/java/bijava-faut-il-casser-la-compatibilite/
Lien6 : http://thierry-leriche-dessirier.developpez.com/tutoriels/java/tuto-google-collections/fichiers/tuto-google-col.zip
Lien7 : http://code.google.com/p/google-collections/
Lien8 : http://code.google.com/p/guava-libraries/
Lien9 : http://google-collections.googlecode.com/svn/trunk/javadoc/com/google/common/collect/Collections2.html
Lien10 : http://google-collections.googlecode.com/svn/trunk/javadoc/com/google/common/collect/Iterables.html
Lien11 : http://google-collections.googlecode.com/svn/trunk/javadoc/com/google/common/base/Predicates.html
Lien12 : http://thierry-leriche-dessirier.developpez.com/tutoriels/java/tuto-google-collections/
Lien13 : http://www.editions-eni.fr/Livres/Les-EJB-3-avec-Struts-2--JSF-2--JasperReports-3--Flex-3-Developpez-pour-le-web-par-l-exemple--3-applications-detaillees/.6_3a6222cf-b921-41f5-886c-c989f77ba994_75c514cc-b7b3-49ed-a7ca-c8e516ab7429_e5bf3eda-7ad1-4a29-a943-f93cf015b594_1_0_d9bd8b5e-f324-473f-b1fc-b41b421c950f.html
Lien14 : http://java.developpez.com/livres/
Lien15 : http://framework.zend.com/
Lien16 : http://oss.oetiker.ch/mrtg/
Lien17 : http://maxime-varinard.developpez.com/tutoriels/php/benchmark-zf-cache/
Lien18 : http://php.developpez.com/livres/
Lien19 : http://www.yoyodesign.org/doc/w3c/css2/selector.html
Lien20 : http://debray-jerome.developpez.com/demos/selecteurs.html#selecteurs1
Lien21 : http://debray-jerome.developpez.com/demos/selecteurs.html#selecteurs2
Lien22 : http://debray-jerome.developpez.com/demos/selecteurs.html#selecteurs3
Lien23 : http://debray-jerome.developpez.com/demos/selecteurs.html#combinateur
Lien24 : http://debray-jerome.developpez.com/demos/selecteurs.html#nth-child
Lien25 : http://debray-jerome.developpez.com/demos/selecteurs.html#nth-last-child
Lien26 : http://debray-jerome.developpez.com/articles/les-selecteurs-en-css3/
Lien27 : http://debray-jerome.developpez.com/demos/couleur.html#hsl
Lien28 : http://debray-jerome.developpez.com/demos/couleur.html#hsla
Lien29 : http://debray-jerome.developpez.com/demos/couleur.html#rgba
Lien30 : http://debray-jerome.developpez.com/demos/couleur.html#opacity
Lien31 : http://debray-jerome.developpez.com/articles/comprendre-les-couleurs-en-css3/
Lien32 : http://plambert.developpez.com/tutoriel/css/coins-arrondis/fichiers/image-coulissante.html
Lien33 : http://www.svay.com/coins_arrondis/coins_arrondis.html
Lien34 : http://biboo.net/css-blocs-bords-arrondis
Lien35 : http://www.clb56.fr/test_css/test_roundcorner/
Lien36 : http://www.css3.info/preview/rounded-border/
Lien37 : http://www.roundedcornr.com/
Lien38 : http://plambert.developpez.com/tutoriel/css/coins-arrondis/
Lien39 : http://fray.developpez.com/articles/air/sqlite/
Lien40 : http://fray.developpez.com/tutoriels/air/application-multiplateforme-sqlite/
Lien41 : http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/localDatabaseSQLSupport.html#dataTypes
Lien42 : http://www.adobe.com/fr/products/air/
Lien43 : http://www.developpez.net/forums/f154/bases-donnees/autres-sgbd/sqlite/
Lien44 : http://www.developpez.net/forums/f755/webmasters-developpement-web/flash-flex/flex/
Lien45 : http://fray.developpez.com/articles/air/sqlite/
Lien46 : http://farscape.developpez.com/Articles/VisualStudio2010/
Lien47 : http://blog.developpez.com/farscape/p8879/visualc-mfc/visual-c-c-0x-utilisation-de-nullptr/
Lien48 : http://msdn.microsoft.com/en-us/library/dd293588.aspx
Lien49 : http://blog.developpez.com/farscape/p8882/visualc-mfc/developpement/visual-c-2010-c-0x-utilisation-de-static/
Lien50 : http://msdn.microsoft.com/en-us/library/dd293668.aspx
Lien51 : http://blog.developpez.com/farscape/p8895/visualc-mfc/visual-c-2010-c-0x-utilisation-des-refer/
Lien52 : http://msdn.microsoft.com/en-us/library/dd293667.aspx
Lien53 : http://blog.developpez.com/farscape/p8903/visualc-mfc/developpement/visual-c-2010-c-0x-utilisation-du-mot-cl/
Lien54 : http://msdn.microsoft.com/en-us/library/dd537655.aspx
Lien55 : http://blog.developpez.com/farscape/p8906/visualc-mfc/developpement/visual-c-2010-c-0x-utilisation-de-delcty/
Lien56 : http://blog.developpez.com/farscape/p8925/visualc-mfc/visual-c-2010-c-0x-utilisation-du-traili/
Lien57 : http://msdn.microsoft.com/fr-fr/library/dd293603.aspx
Lien58 : http://blog.developpez.com/farscape/p9281/visualc-mfc/developpement/visual-c-2010-c-0x-utilisation-des-expre/
Lien59 : http://qt.nokia.com/qtdevdays2010
Lien60 : http://qt.developpez.com/evenement/2010-devdays/
Lien61 : http://www.developpez.net/forums/d963194/c-cpp/bibliotheques/qt/qt-devdays-2010-participation-developpez-com-presentation-conferences/
Lien62 : http://qt.nokia.com/
Lien63 : http://qt-quarterly.developpez.com/qq-33/mars-venus/
Lien64 : http://qt.nokia.com/downloads
Lien65 : http://qt.nokia.com/whatsnew
Lien66 : http://www.developpez.net/forums/d965034/c-cpp/bibliotheques/qt/developpeurs-viennent-mars-designers-venus/
Lien67 : http://qt-quarterly.developpez.com/
Lien68 : http://www.developpez.net/forums/d825667-2/c-cpp/bibliotheques/qt/qt-creator-2-1-montre-premiere-beta/#post5482412
Lien69 : http://qt.developpez.com/tutoriels/
Lien70 : http://qt.developpez.com/faq/
Lien71 : http://qt.developpez.com/outils/
Lien72 : http://qt.developpez.com/livres/
Lien73 : http://blog.developpez.com/recap/qt/
Lien74 : http://qt.developpez.com/binaires/

Lien75 : http://qt.developpez.com/doc/4.6/index/
Lien76 : http://qt-quarterly.developpez.com/
Lien77 : http://qt-labs.developpez.com/
Lien78 : http://www.developpez.net/forums/d970522/c-cpp/bibliotheques/qt/traductions-qt-labs-lieu-dexperimentation-excellence-developpeurs-qt/
Lien79 : http://doc.qt.nokia.com/4.6/qglwidget.html
Lien80 : http://doc.qt.nokia.com/4.6/qglwidget.html#paintGL
Lien81 : http://doc.qt.nokia.com/4.6/qglwidget.html#resizeGL
Lien82 : http://doc.qt.nokia.com/4.6/qglwidget.html#initializeGL
Lien83 : http://doc.qt.nokia.com/4.6/qtimer.html
Lien84 : http://doc.qt.nokia.com/4.6/qwidget.html#windowTitle-prop
Lien85 : http://doc.qt.nokia.com/4.6/qtimer.html#timeout
Lien86 : http://doc.qt.nokia.com/4.6/qwidget.html#keyPressEvent
Lien87 : http://doc.qt.nokia.com/4.6/qkeyevent.html
Lien88 : http://fhalgand.developpez.com/tutoriels/qt/integration-opengl/01-premiere-application/
Lien89 : http://qt.developpez.com/tutoriels/edi/xcode/integration-de-qt-a-xcode-3/
Lien90 : http://www.eyrolles.com/Informatique/Livre/9782212127751?societe=developpez
Lien91 : http://e-sarrion.developpez.com/cours/dev-web-mobile/bases-html/
Lien92 : http://mac.developpez.com/livres/
Lien93 : http://www.developpez.net/forums/d980815/club-professionnels-informatique/actualites/virus-stuxnet-complexite-precedent-attaque-infrastructures-industrielles-liran/
Lien94 : http://www.developpez.net/forums/d924808/club-professionnels-informatique/actualites/reseau-social-open-source-diaspora-libere-code-jour-lancement-cours-mois-doctobre/
Lien95 : http://msdn.microsoft.com/en-us/library/bb545671%28v=VS.85%29.aspx
Lien96 : http://msdn.microsoft.com/en-us/library/bb530716%28v=VS.85%29.aspx
Lien97 : http://ram-0000.developpez.com/tutoriels/cpp/Windows-Privileges/#L2.3
Lien98 : http://www.microsoft.com/downloads/details.aspx?FamilyID=9d467a69-57ff-4ae7-96ee-b18c4790cffd&displaylang=en
Lien99 : http://ram-0000.developpez.com/tutoriels/cpp/Windows-Privileges/images/Privileges.exe
Lien100 : http://ram-0000.developpez.com/tutoriels/cpp/Windows-Privileges/images/Privileges.zip
Lien101 : http://ram-0000.developpez.com/tutoriels/Commun/vcredist_x86_8.0.50727.4053.exe
Lien102 : http://msdn.microsoft.com/en-us/library/aa379295%28VS.85%29.aspx
Lien103 : http://msdn.microsoft.com/en-us/library/aa446671%28VS.85%29.aspx
Lien104 : http://msdn.microsoft.com/en-us/library/aa379180%28VS.85%29.aspx
Lien105 : http://msdn.microsoft.com/en-us/library/aa379176%28v=VS.85%29.aspx
Lien106 : http://msdn.microsoft.com/en-us/library/aa379168%28v=VS.85%29.aspx
Lien107 : http://msdn.microsoft.com/en-us/library/aa375202%28VS.85%29.aspx
Lien108 : http://ram-0000.developpez.com/tutoriels/cpp/Windows-Privileges/#L4.5
Lien109 : http://ram-0000.developpez.com/tutoriels/cpp/Windows-Privileges/
Lien110 : http://securite.developpez.com/livres/
Lien111 : http://alpha-arts.net/projets.html
Lien112 : http://www.sfml-dev.org/
Lien113 : http://www.youtube.com/watch?v=43JtypUL3BY&feature=player_embedded
Lien114 : http://www.alpha-arts.net/blog/articles/view/18/moteur-de-lumieres
Lien115 : http://holyspirit.alpha-arts.net/Autres/Holyspirit%20Light%20Effect%20Test%20NEW.rar
Lien116 : http://holyspirit.alpha-arts.net/?p=dljeu
Lien117 : http://gregouar.developpez.com/tutoriels/jeux/moteur-lumieres-dynamiques-2d/
Lien118 : http://www.opengl.org/registry/
Lien119 : http://nehe.developpez.com/tutoriel/01-introduction/
Lien120 : http://tbayart.developpez.com/articles/opengl/fenetrage/sdl/
Lien121 : http://jeux.developpez.com/tutoriels/cours-opengl/
Lien122 : http://www.opengl.org/registry/
Lien123 : http://www.opengl.org/registry/specs/ARB/half_float_pixel.txt
Lien124 : http://www.opengl.org/registry/specs/ARB/vertex_shader.txt
Lien125 : http://www.opengl.org/registry/specs/NV/texture_shader2.txt
Lien126 : http://www.opengl.org/registry/specs/EXT/vertex_array_bgra.txt
Lien127 : http://www.opengl.org/sdk/docs/man/xhtml/glGetString.xml
Lien128 : http://www.opengl.org/sdk/docs/man/xhtml/glGetError.xml
Lien129 : http://www.opengl.org/registry/api/glext.h
Lien130 : http://www.opengl.org/registry/api/gl3.h
Lien131 : http://glew.sourceforge.net/
Lien132 : http://glew.sourceforge.net/basic.html
Lien133 : http://www.opengl.org/sdk/libs/GLee/
Lien134 : http://www.opengl.org/
Lien135 : http://www.khronos.org/
Lien136 : http://www.opengl.org/resources/features/OGLextensions/
Lien137 : http://www.opengl.org/registry/
Lien138 : http://glew.sourceforge.net/index.html
Lien139 : http://www710.univ-lyon1.fr/%7Ejciehl/Public/OpenGL_PG/apc.html
Lien140 : http://alexandre-laurent.developpez.com/tutoriels/OpenGL/OpenGL-Extensions/