



Develloppez

Le Mag

Édition d'avril – mai 2015

Numéro 57

Magazine en ligne gratuit

Diffusion de copies conformes à l'original autorisée

Réalisation : Alexandre Pottiez

Rédaction : la rédaction de Develloppez

Contact : magazine@redaction-developpez.com

Sommaire

C++	Page	2
JavaScript	Page	6
Java	Page	20
Eclipse	Page	29
Android	Page	46
Programmation	Page	53
2D/3D/Jeux	Page	68
Qt	Page	71
Access	Page	82
Excel	Page	99

Éditorial

Le magazine revient avec toujours plus d'articles pour satisfaire votre envie de connaissance.

La rédaction

Article Programmation



Le débogage d'une application : méthodes et exercices

Vous êtes face à un bogue et vous ne savez pas où il se trouve. On vous dit d'utiliser un « débogueur ». Vous en avez un, mais vous ne l'avez jamais utilisé et vous ne savez pas quoi faire. Cet article est pour vous!

par [Alexandre Laurent](#)

Page 53



Article 2D/3D/Jeux

Vulkan, la nouvelle bibliothèque de hautes performances pour le GPU

Découvrez la nouvelle bibliothèque pour le GPU de Khronos.

Celle-ci se place comme successeur d'OpenGL.

par [Alexandre Laurent](#)

Page 68

C++



Les derniers tutoriels et articles

Métaprogrammation et métafonctions en C++11

1 Introduction

Les fonctions et classes template en C++03 ainsi que la possibilité de les spécialiser, ont permis la création d'outils puissants. L'un des plus importants reste sans doute la bibliothèque Boost.MPL ([lien 1](#)), un framework de métaprogrammation de template de haut niveau.

Tout le monde n'a pas besoin de connaître ou d'utiliser la métaprogrammation pour être un bon programmeur, mais l'outil peut sembler trop théorique. Cependant, il y a des domaines où la métaprogrammation est utile : par exemple, les traits ou les analyses dimensionnelles ([lien 2](#)) effectuées à la compilation.

En revanche, l'emploi de ces techniques n'est pas toujours conseillé pour la simple raison qu'elles sont très difficiles à utiliser pour le programmeur lambda. La faute en incombe à la conception du C++03 qui n'était pas très orientée vers le support de la métaprogrammation qui, de plus, a été découverte il y

a peu, d'où l'utilisation peut-être excessive du principe des templates.

Un code qui utilise la métaprogrammation n'est pas facile à lire ni à écrire ; en contrepartie, son emploi permet quelque chose de fantastique : effectuer des calculs et utiliser leur résultat au moment de la compilation !

Heureusement, la norme C++11 a apporté son lot d'outils pour la rendre plus abordable et facile d'accès. De plus en plus de programmeurs peuvent utiliser ces techniques sans passer des nuits blanches à comprendre la mécanique des templates ou à analyser de mystérieux messages d'erreur du compilateur. Dave Abrahams a participé à un débat sur le thème de la « métaprogrammation en C++11 » ([lien 3](#)) lors de la conférence C++ Now! de 2012, à Aspen, dans le Colorado.

Voici maintenant une petite introduction sur deux concepts de base de la métaprogrammation.

2 Qu'est-ce qu'une métafonction ?

Tout d'abord, que signifie le préfixe « méta » ici ? Ma définition est que c'est un bout de code pouvant être exécuté au cours de la compilation du programme et son résultat utilisé à ce moment. Contrai-

rement aux fonctions « classiques » qui se lancent normalement durant l'exécution du programme, une métafonction peut renvoyer deux choses : une valeur ou un type.

3 Calcul des valeurs

Le calcul des valeurs à la compilation en C++03 est basé uniquement sur le comportement des templates qui, en dehors des types, peuvent aussi être paramétrés avec des valeurs entières. Par exemple :

```
1 template<int i>
2 struct IntArray
3 {
4     int value[i];
5 };
6
7 template<int i, int j>
8 struct Multiply
```

```
9 {
10     static const int value = i * j;
11 }
```

La structure IntArray nous montre que les paramètres de template non typés ont réellement été ajoutés pour le C++03.

La structure Multiply nous montre un exemple simple de métafonction : on donne deux valeurs de type int et nous pouvons en trouver une troisième pendant la compilation.

```
1 int array[ Multiply<3, 4>::value ]; //
   un array
```

La formule `Multiply<3, 4>::value` ressemble difficilement à un appel de fonction, mais c'en est pourtant un : vous entrez des valeurs et vous obtenez un résultat.

C'est ici que les questions intéressantes commencent à arriver...

Savez-vous que chaque métafonction est déclarée comme une classe artificielle et qu'elle est évaluée en instanciant une classe template et en accédant à un membre statique ? Si vous travaillez depuis un moment avec les métafonctions et que vous êtes habitué(e) à leur syntaxe, vous pouvez ne pas réaliser que ça puisse être un problème.

Si vous pensez que ce premier exemple est simple, passons à un exemple un peu plus difficile.

Calculons une factorielle, pour voir. Je ne pense pas que quiconque ait besoin de calculer une factorielle durant la compilation, mais c'est l'exemple le plus accessible que je puisse vous présenter pour illustrer les concepts de base de la programmation fonctionnelle : la récursivité, les conditions d'arrêt de récursivité et la gestion des erreurs.

```
1 template<int i> struct Factorial
2 {
3     static const int value = i *
        Factorial<i - 1>::value;
4 };
5
6 template<> struct Factorial<0>
7 {
8     static const int value = 1;
9 };
```

La première définition introduit la récursion. La valeur `value` de l'instanciation courante est déterminée par la valeur correspondante d'une autre instanciation. La seconde définition correspond à la condition finale de récursivité, pour `i == 0`.

Une autre chose à voir, c'est la vérification d'une précondition pour signaler une erreur le cas échéant. Dans le cas de la factorielle, on veut éliminer la possibilité d'avoir des arguments négatifs. Bien sûr, on aurait pu utiliser le type `unsigned int` pour éviter le problème, mais le but de cet exercice est de montrer comment la vérification d'une précondition peut être implémentée en général.

Ça nécessite de définir d'autres métafonctions en amont de ce bloc vulnérable.

```
1 template<int i, bool c> struct
   NegativeArgument
2 {
3     static const int value = Factorial<i
        >::value;
4 };
5
6 template<int i> struct NegativeArgument<
   i, false>; //indéfini
7
8 template<int i> struct SafeFactorial
9 {
```

```
10     static const int value =
        NegativeArgument<i, (i >= 0)>::
        value;
11 };
```

Prenons l'exemple en partant de la fin. La métafonction `SafeFactorial` transmet l'argument à une autre métafonction : `NegativeArgument`, mais elle passe aussi par la précondition (`i >= 0`), pour éviter les cas où `i` est négatif.

Le nom de la métafonction suivante peut sembler obscur à ce stade, mais il vous paraîtra beaucoup moins abstrait quand il s'agira de générer des messages d'erreur à la compilation.

La clé de voûte de cet exemple se trouve dans la métafonction `NegativeArgument`. La spécification pour le booléen à `false` est déclarée explicitement (pour intercepter un non-respect de la précondition) mais laissée indéfinie pour être sûr que le code qui utilise cette spécialisation va provoquer une erreur à la compilation. S'il y a une erreur lors de la compilation, nous allons avoir un message d'erreur disant quelque chose comme « utilisation de type non défini `NegativeArgument<i,c>` ». Ce n'est pas un message parfaitement explicite, mais je l'espère suffisamment clair pour vous donner une idée de ce qui ne va pas.

La version à deux paramètres de `NegativeArgument` va directement transmettre l'argument à notre bonne métafonction `Factorial`.

Le point que j'essaie de mettre en lumière ici est que la définition de métafonctions comme celle-là est un peu compliquée, voire ésotérique dans certains cas.

Comment C++11 va-t-il nous aider ? En étendant le concept des expressions constantes. Vous avez probablement déjà entendu parler de `constexpr` et ce qu'il vous permet de faire.

Pour faire court, en C++11, nos deux versions (sécurisée et non sécurisée) du calcul de factorielle durant le temps de compilation peuvent être définies ainsi :

```
1 constexpr int factorial(int i)
2 {
3     return (i == 0) ? //
        condition terminale
4         1 : // et
        valeur terminale
5         i * factorial(i - 1); // dé
        finition de la récursivité
6 }
7
8 constexpr int safe_factorial(int i)
9 {
10     return (i < 0) ? //
        condition d'erreur
11         throw exception() : // géné
        ration d'erreur (
        compilation)
12         factorial(i); // vrai
        calcul
13 }
```

Une fonction constexpr est presque comme une fonction normale, mais nous devons utiliser un opérateur de condition plutôt qu'un if. Grâce à cet outil, nous pouvons déclarer un tableau de 24 éléments de cette façon :

```
1 int array[ factorial(4) ];
```

Encore une fois, j'ai choisi un exemple de calcul de factorielle, car c'est une opération très simple à

4 Calcul de types

Un autre type de métafonction est celui où nous allons passer un type comme argument et en avoir un autre en retour. Pour exemple, essayons d'implémenter la fonction `remove_pointer` (comme celle de la STL) qui, pour les types $U = T^*$, retourne T et qui, pour tous les autres types, retourne le même type inchangé. Autrement dit, la métafonction supprime le pointeur de plus haut niveau là où il est possible de l'enlever.

C'est possible grâce à une spécialisation partielle de template :

```
1 template<typename U> // en général
2 struct remove_pointer
3 {
4     typedef U type;
5 };
6
7 template<typename T> // pour U = T*
8 struct remove_pointer<T*>
9 {
10     typedef T type;
11 };
```

Cela ne semble pas si mal, mais dans l'état actuel des choses, pour utiliser notre métafonction dans un template de fonction, vous devrez utiliser une syntaxe quelque peu étrange :

```
1 template<typename T>
2 typename remove_pointer<T>::type fun(T
    val);
```

La nécessité d'utiliser le peu pratique `typename` est une conséquence des règles de la spécialisation partielle des classes template. Le compilateur a besoin d'être préparé aux vilaines spécialisations comme celle qui suit :

```
1 template<typename U> // template ma
    itre
2 struct MyClass
3 {
4     typedef U type; // définition
    d'un type
5 };
6
7 template<typename T> // spé
    cialisation
8 struct MyClass<const T>
9 {
10     static int type = 0; // définition
    d'un objet
11 };
```

définir et que cet exemple tient facilement dans cet article. Vous n'aurez probablement jamais de votre vie à calculer une factorielle pendant la compilation. Mais il est tout à fait possible de devoir calculer un jour le plus grand commun diviseur si on souhaite implémenter une bibliothèque de nombres rationnels durant la compilation.

Vous pouvez avancer qu'un compilateur doit être assez intelligent pour pouvoir faire ce travail sans l'aide du programmeur, mais ce n'est pas facile à implémenter en général et... c'est simplement ainsi que le C++ fonctionne. À la fin, la perspective d'avoir à écrire du code comme `typename remove_pointer<T>::type` est peu attirante et rend le code difficile à lire, surtout pour vos collègues qui ne veulent pas être embêtés par ce genre de chose, mais tel est le C++03.

Heureusement, le C++11 offre un certain confort : les alias de templates (lien 4). Les alias de type sont une nouvelle forme pour définir des types, un peu à la manière de `typedef`, mais avec une syntaxe améliorée :

```
1 using Integer = int;
2 // même chose que : typedef int Integer;
3
4 using FunPtr = int* (*)(int*);
5 // même chose que : typedef int*(*FunPtr
    )(int*);
```

Les alias de templates ajoutent une fonctionnalité qui manquait depuis longtemps au C++03 :

```
1 template<typename T>
2 using StackVector = std::vector<T,
    StackAllocator<T>>;
3
4 StackVector<int> v;
```

Avec les alias de templates, nous pouvons écrire notre métafonction d'effacement de pointeur comme suit :

```
1 template<typename U> // en géné
    ral
2 struct remove_pointer
3 {
4     using type = U;
5 };
6
7 template<typename T> // pour U =
    T*
8 struct remove_pointer<T*>
9 {
10     using type = T;
11 };
12
13 template<typename W>
14 using RemovePointer = typename
    remove_pointer<W>::type;
```

La définition de notre métafonction n'est pas plus courte. Aussi, nous avons toujours besoin d'utiliser l'horrible typename. Cependant, la façon dont la métafonction est utilisée a été améliorée significativement :

```
1 template<typename T>
```

```
2 RemovePointer<T> fun(T val);
```

Cette technique pour noter les métafonctions a été employée dans le rapport « A Concept Design for the STL » par B. Stroustrup et A. Sutton (lien 5) ainsi que dans les bibliothèques Origin (lien 6).

5 Essayez vous-même

Il y a plusieurs fonctionnalités qui rendent la métaprogrammation plus simple en C++11, les assertions statiques en font partie (lien 7). Quelques évolutions viennent simplement de l'amélioration des implémentations par le compilateur (à savoir un meilleur support pour l'instanciation des tem-

plates récursifs). J'ai seulement listé les deux que je trouvais les plus intéressantes. Les expressions constantes généralisées (constexpr) sont implémentées dans GCC depuis la version 4.6 tout comme les alias de templates ont été introduits dans GCC 4.7 et Clang 3.0.

Retrouvez l'article de **Andrzej Krzemiński** traduit par **Guy Arbus** en ligne : [lien 8](#)



JavaScript

Les derniers tutoriels et articles

Le jeu Mario5 avec TypeScript

De JavaScript en TypeScript, la conversion par l'exemple

Revue des principaux axes de travail pour convertir un code JavaScript en code TypeScript idiomatique.

1 Avant-propos

Ce qui suit est une traduction d'un article de Florian Rappil dont le but premier est de montrer les principaux axes de travail pour transformer du code JavaScript en code TypeScript idiomatique.

Pour cela, l'auteur se base sur un de ses projets, le jeu Mario5 initialement écrit en JavaScript et sur lequel il a déjà écrit un article concernant la conception et l'implémentation du jeu. Ce n'est pas ce qui

sera abordé ici, mais uniquement la conversion en TypeScript à proprement parler.

Bien qu'il puisse être utile de lire au préalable l'article dédié au développement du jeu Mario5, cela reste facultatif dans la mesure où les divers aspects de la conversion sont illustrés à la fois par du code JavaScript issu du jeu initial et par du code TypeScript résultant de la conversion.

2 Introduction

En ce qui me concerne, l'un des moments les plus mémorables sur CodeProject a été la publication de l'article sur Mario5 : [lien 9](#). Dans l'article, je décrivais la réalisation d'un jeu basé sur les technologies Web comme HTML5, CSS3 et JavaScript. L'article a eu pas mal de succès et est probablement l'un de ceux dont je suis vraiment fier.

L'article original utilise quelque chose que j'ai décrit comme du « JavaScript orienté objet ». J'ai écrit une petite bibliothèque auxiliaire appelée *oop.js*, qui m'a permis d'utiliser l'héritage inspiré du modèle des classes. Évidemment, JavaScript est très orienté objet depuis ses débuts. Les classes ne sont pas uniquement une caractéristique essentielle de la POO. Néanmoins, ce modèle m'a été très utile pour rendre le code à la fois, facile à lire et à maintenir. Et finalement, ceci permet de ne pas avoir à traiter directement avec le concept de prototype.

Avec TypeScript nous avons à disposition une construction normalisée des classes en JavaScript. La syntaxe est basée sur la version ES6, faisant de TypeScript un surensemble de JavaScript ES5 et

prochainement de ES6. Bien sûr, TypeScript transpile en ES3 ou ES5, ce qui signifie que les classes seront décomposées en quelque chose qui est accepté dès maintenant : les prototypes. Néanmoins, ce qui reste est un code qui est lisible, compatible ES3 ou ES5, fiable et partageant une base commune. Avec mon approche spécifique (*oop.js*), personne d'autre que moi ne pouvait dire ce qu'il se passait sans lire le code de ma bibliothèque auxiliaire. Avec TypeScript, un large éventail de développeurs utilisent le même modèle, car il est intégré dans le langage.

Cela coulait donc de source de convertir le projet Mario5 en TypeScript. Pourquoi en faire un article ? Je pense que c'est un excellent cas pratique sur la manière de convertir un projet. Il illustre également les principaux aspects de TypeScript. Et enfin, il donne une bonne introduction à sa syntaxe et à son comportement. Après tout, TypeScript est simple pour ceux qui connaissent déjà JavaScript et facilite l'apprentissage de JavaScript, pour ceux qui n'en ont pas encore l'expérience.

3 Contexte

Il y a plus d'un an, Anders Hejlsberg de chez Microsoft annonçait un nouveau langage appelé TypeScript. Il était surprenant pour la plupart des gens que Microsoft (et surtout Anders) aille à l'encontre des langages dynamiques, en particulier JavaScript. Cependant, il s'est avéré que Microsoft a compris l'opportunité que représentait sa transition de la programmation à usage général vers la programmation Web. Avec JavaScript au sein des applications Windows Store, l'engouement actuel pour node.js et le mouvement NoSQL avec les bases de données orientées documents utilisant JavaScript pour l'exécution de requêtes, il est évident que JavaScript est aujourd'hui central.

Avoir compris cela a influencé la conception d'un nouveau langage. Au lieu de créer un nouveau langage à partir de zéro (comme Google l'a fait avec Dart), Anders a décidé qu'un nouveau langage se devait d'étendre JavaScript. Aucune solution ne devant être orthogonale. Le problème avec CoffeeScript est qu'il masque JavaScript. Cela peut être attrayant pour certains développeurs, mais pour la plupart d'entre eux, c'est un critère d'exclusion absolu. Anders a décidé que ce langage devait être fortement typé, même si seul le compilateur (ou transpileur pour être plus correct) voit ces annotations.

Qu'est-il donc arrivé ? Un véritable surensemble de ECMAScript 5 a été créé. Ce surensemble a été appelé TypeScript pour indiquer son lien étroit avec JavaScript (ou ECMAScript en général), avec des annotations supplémentaires de type. Toutes les autres fonctionnalités, comme les interfaces, les énumérations, les types génériques, les conversions ex-

plicités, etc. découlent de ces annotations de type. À l'avenir, TypeScript évoluera. Principalement dans deux domaines :

1. Englober ES6 afin de rester un véritable sur-ensemble de JavaScript ;
2. Apporter de nouvelles fonctionnalités pour rendre plus facile le développement en JS.

Il y a principalement deux avantages à l'utilisation de TypeScript. Le premier aspect est que nous pouvons être informé des erreurs et des problèmes potentiels lors de la compilation. Si un argument n'est pas conforme à sa signature, alors le compilateur renvoie une erreur. Cela est particulièrement utile lorsque vous travaillez avec des équipes ou des projets de taille importante. Le second aspect est également intéressant. Microsoft est connue pour son outillage de très bonne facture avec Visual Studio. Mais fournir un bon outillage au langage JavaScript est délicat en raison de sa nature dynamique. Par conséquent, la moindre refactorisation, même simple, comme renommer une variable, peut ne pas être effectuée avec la fiabilité souhaitée.

TypeScript nous fournit un support important au niveau de l'outillage combiné avec une bien meilleure compréhension sur la façon dont notre code va fonctionner. La combinaison de la productivité et de la robustesse est l'argument le plus attrayant pour utiliser TypeScript. Dans cet article, nous allons explorer comment convertir un projet existant. Nous allons voir que la transformation d'un code en TypeScript peut se faire progressivement.

4 Convertir un projet existant

TypeScript ne cache pas le JavaScript. Il commence à partir du JavaScript de base.



La première étape dans l'utilisation de TypeScript est bien sûr d'avoir des fichiers source TypeScript. Puisque nous voulons utiliser TypeScript dans un projet existant, nous devons convertir ces fichiers. Il n'y a pas de condition préalable si ce n'est de renommer nos fichiers **.js* en **.ts*. C'est juste une question de convention, qui n'est pas réellement

nécessaire. Néanmoins, comme le compilateur TypeScript **tsc** considère généralement en entrée les fichiers **.ts*, et en sortie des fichiers **.js*, renommer l'extension garantit que rien de fâcheux ne se passera.

Les prochains paragraphes traitent des améliorations progressives dans le processus de conversion. Nous supposons à présent que chaque fichier a l'extension TypeScript usuelle **.ts*, même si aucune fonctionnalité spécifique à TypeScript n'est encore utilisée.

4.1 Référencement

La première étape est de fournir dans un fichier JavaScript les références de tous les autres fichiers JavaScript qui lui sont nécessaires. Habituellement, nous n'avons qu'à gérer des fichiers indépendants qui cependant (généralement) doivent être insérés dans

un certain ordre dans notre code HTML. Les fichiers JavaScript ne connaissent pas le fichier HTML, ils ne connaissent pas leur ordre dans le fichier HTML (sans même parler de savoir quels fichiers JavaScript sont insérés).

Comme nous voulons donner quelques indications à notre compilateur intelligent (TypeScript), nous devons préciser que d'autres éléments peuvent être disponibles. Par conséquent, nous devons ajouter des références au début des fichiers source. Ces références précisent tous les autres fichiers qui seront utilisés dans le fichier en cours.

Par exemple, nous pourrions inclure jQuery (utilisé par exemple dans le fichier *main.ts*) par sa définition via :

```
1 ///
```

Nous pourrions également inclure une version TypeScript de la bibliothèque ou bien sa version JavaScript même s'il est préférable de n'inclure que le fichier de définition. Les fichiers de définition ne contiennent aucune logique. Cela rend ces fichiers significativement plus petits et plus rapides à analyser. En outre, ces fichiers contiennent généralement davantage de commentaires et sont de meilleure qualité. Enfin, alors que nous pourrions préférer nos propres fichiers **.ts* aux fichiers **.d.ts*, il faut savoir que pour le cas de jQuery et pour d'autres bibliothèques, elles sont à l'origine écrites en JavaScript et auront besoin d'un fichier de définition pour que tout fonctionne correctement.

Nous pouvons être amenés à décrire par nous-mêmes des fichiers de définition simples. L'exemple le plus basique est le fichier *def/interfaces.d.ts*. Celui-ci ne contient pas de code à proprement parler, par conséquent une compilation de ce fichier serait inutile. Le référencement de ce fichier, par contre, est logique, puisque les informations supplémentaires fournies sur les types par ce fichier nous aident à annoter notre code.

4.2 Annotations

La caractéristique la plus importante de TypeScript est les annotations de type. En fait, le nom de ce langage indique la grande importance de cette fonctionnalité.

La plupart des annotations de type ne sont pas réellement nécessaires. Si une variable est immédiatement affectée (nous définissons une variable, au lieu de simplement la déclarer), alors le compilateur peut déduire le type de la variable.

```
1 var basepath = 'Content/';
```

Évidemment, le type de cette variable est une chaîne (string). C'est aussi ce que déduit TypeScript. Néanmoins, on pourrait aussi mentionner le type explicitement.

```
1 var basepath: string = 'Content/';
```

En général, il n'est pas recommandé d'être explicite sur ces annotations. Cela encombre inutilement le code et le rend moins souple. Cependant, parfois, ces annotations sont nécessaires. Bien sûr, le cas le plus évident est lorsque nous ne faisons que déclarer une variable :

```
1 var frameCount: number;
```

Il y a d'autres scénarios. Envisageons la création d'un objet simple pouvant être étendu avec des propriétés supplémentaires. Le code JavaScript habituel ne contient pas assez d'information pour le compilateur :

```
1 var settings = { };
```

Quelles sont les propriétés disponibles ? Quel est le type de ces propriétés ? Peut-être que nous ne le savons pas et nous voulons utiliser cet objet comme un dictionnaire. Dans ce cas, nous devrions spécifier le cadre d'utilisation de cet objet :

```
1 var settings: any = { };
```

Mais il y a aussi un autre cas. Nous savons déjà quelles propriétés peuvent être disponibles, et nous avons seulement besoin de définir ou de récupérer une partie de ces propriétés optionnelles. Dans ce cas, on peut tout à fait spécifier le type exact :

```
1 var settings: Settings = { };
```

Le cas le plus important a été omis jusqu'ici. Tandis que les variables (locales ou globales) peuvent être déduites dans la plupart des cas, les paramètres d'une fonction ne peuvent jamais être déduits. En toute rigueur, les paramètres d'une fonction peuvent être déduits dans une seule situation (comme avec les types de paramètres génériques), mais pas dans la fonction elle-même. Nous devons donc dire au compilateur le type des paramètres que nous avons.

```
1 setPosition(x: number, y: number) { this  
    .x = x; this.y = y; }
```

Transformer JavaScript de façon incrémentale avec les annotations de type est donc un processus qui commence par la mise à jour de la signature des fonctions. Dans ce cas, quelles sont les bases concernant ces annotations ? Nous avons déjà appris que `number`, `string` et `any` sont des types primitifs, qui représentent des types élémentaires. De plus, nous avons `boolean` et `void`. Ce dernier n'est utile que pour le type de retour des fonctions. Il indique que rien d'utile n'est retourné (les fonctions JS retournent toujours quelque chose, par défaut `undefined`).

Qu'en est-il des tableaux ? Un tableau standard est de type `any[]`. Si nous voulons indiquer que seuls les nombres peuvent être utilisés avec ce tableau, nous pourrions l'annoter `number[]`. Les tableaux multidimensionnels sont aussi possibles. Une matrice peut être annotée comme `number[][]`. En raison de la nature de JavaScript, nous n'avons que des tableaux irréguliers pour les dimensions multiples.

4.3 Énumérations

Maintenant que nous avons commencé à annoter nos fonctions et nos variables, il va nous falloir des types personnalisés. Bien sûr, nous avons déjà quelques types ici et là. Cependant, ces types peuvent ne pas être aussi précis que nous le souhaiterions, ou bien être définis de façon trop spécifique.

TypeScript peut offrir de meilleurs choix. Les collections de constantes numériques, par exemple, peuvent être définies comme une énumération. Dans l'ancien code nous avions des objets tels que :

```
1 var directions = { none: 0, left: 1, up: 2, right: 3, down: 4 };
```

Il n'est pas évident que les éléments contenus soient des constantes. Ils pourraient facilement être modifiés. Un compilateur ne pourrait-il pas générer une erreur si nous faisons des choses inappropriées avec un tel objet ? C'est là que le type enum est utile. Actuellement, ils sont limités à des nombres, cependant, pour la plupart des collections cela est suffisant. Plus important encore, ils sont considérés comme des types, ce qui signifie que nous pouvons les utiliser dans nos annotations de type.

Une majuscule a été ajoutée en début de nom, ce qui indique que Direction est en effet un type. Puisque nous ne voulons pas l'utiliser comme une énumération en mode binaire, nous utilisons la version simple (par rapport à la convention de .NET, ce qui est logique dans ce scénario).

```
1 enum Direction { none = 0, left = 1, up = 2, right = 3, down = 4, };
```

Maintenant, nous pouvons l'utiliser dans le code ainsi :

```
1 setDirection(dir: Direction) { this.direction = dir; }
```

Notez que le paramètre dir est annoté de manière à restreindre les arguments au type Direction. Cela exclut les nombres quelconques et impose d'utiliser les valeurs de l'énumération Direction. Que faire si nous avons une entrée de l'utilisateur qui se trouve être un nombre ? Dans un tel scénario, nous pouvons également forcer le type et utiliser une conversion explicite (*cast*) TypeScript :

```
1 var userInput: number; // ...
   setDirection(<Direction>userInput);
```

Les conversions explicites en TypeScript ne fonctionnent que si elles sont légales. Étant donné que chaque Direction est un nombre, un certain nombre pourrait être une Direction valide. Parfois, une conversion explicite peut à l'avance être détectée comme invalide. Si userInput est de type string, TypeScript se plaint et retourne une erreur de conversion.

4.4 Interfaces

Les interfaces définissent des types sans spécifier leur implémentation. Elles disparaîtront complètement dans le code JavaScript résultant, tout comme nos annotations de type. Essentiellement, elles sont assez semblables aux interfaces en C#, cependant, il y a quelques différences notables.

Examinons une interface de notre code :

```
1 interface LevelFormat { width: number; height: number; id: number; background: number; data: string [] []; }
```

Cela définit le format d'une définition d'un niveau du jeu. Nous voyons qu'une telle définition consiste en des nombres tels que la largeur (width), la hauteur (height), l'arrière-plan (background) et un identifiant (id). De plus, une chaîne bidimensionnelle (data) définit les différentes tuiles qui doivent être utilisées dans le niveau.

Nous avons déjà mentionné que les interfaces en TypeScript sont différentes de celles en C#. Une des raisons est que TypeScript autorise la fusion des interfaces. Si une interface a le nom d'une interface déjà existante, celle-ci ne sera pas écrasée. Il n'y aura aucun avertissement du compilateur ni erreur. Au lieu de cela, l'interface existante sera étendue avec les propriétés définies dans la nouvelle.

L'interface suivante fusionne avec l'interface préexistante Math (issue des définitions de base de TypeScript). Ajoutons une nouvelle méthode :

```
1 interface Math { sign(x: number): number ; }
```

Les méthodes sont déclarées en spécifiant les paramètres entre parenthèses. Communément, l'annotation de type concerne le retour de la méthode. Avec l'interface que nous venons d'étendre, le compilateur de TypeScript nous permet d'écrire la méthode suivante :

```
1 Math.sign = function(x: number) { if (x > 0) return 1; else if (x < 0) return -1; return 0; };
```

Une autre option intéressante avec les interfaces TypeScript est la déclaration hybride. En JavaScript, un objet n'est pas limité à être un simple transporteur de paires clé-valeur. Un objet peut aussi être invoqué comme une fonction. Un bon exemple d'un tel comportement est jQuery. Il y a plusieurs façons possibles d'appeler l'objet jQuery, chacune d'entre elles retournant une nouvelle sélection jQuery. De plus, l'objet jQuery comporte également des propriétés qui se révèlent être des outils tout à fait intéressants et utiles.

Dans le cas de jQuery, une des interfaces ressemble à ceci :

```
1 interface JQueryStatic { (): JQuery; (html: string, ownerDocument?: Document): JQuery; ajax(settings:
```

```
jQueryAjaxSettings): JQueryXHR; /*
... */ }
```

Ici, nous avons deux appels possibles (parmi d'autres) et une propriété qui est directement mise à disposition. Les interfaces hybrides nécessitent que l'implémentation soit une fonction enrichie d'autres propriétés.

On peut aussi créer des interfaces basées sur d'autres interfaces (ou des classes, qui seront utilisées comme interfaces dans ce contexte).

Prenons le cas suivant. Pour caractériser les points, nous utilisons l'interface Point. Ici, nous déclarons deux coordonnées x et y. Pour définir une image, nous avons besoin de deux valeurs. Un emplacement (offset), où l'image doit être placée, et la chaîne de caractères qui représente la source de l'image.

Par conséquent, nous pouvons définir une interface comme étant la dérivation/spécialisation de l'interface Point. Le mot-clé extends permet cela en TypeScript.

```
1 interface Point { x: number; y: number;
  } interface Picture extends Point {
  path: string; }
```

Nous pouvons faire dériver d'autant d'interfaces que nous le souhaitons, il suffit pour cela de séparer les noms des interfaces par des virgules.

4.5 Classes

À ce stade, nous avons déjà typé la majeure partie de notre code, mais un concept important n'a pas été traduit en TypeScript. Le code source d'origine introduit le concept de classe (y compris l'héritage) dans JavaScript. Au départ, cela ressemblait à l'exemple suivant :

```
1 var Gauge = Base.extend({ init: function
  (id, startImgX, startImgY, fps,
  frames, rewind) { this._super(0, 0);
  this.view = $('#' + id); this.
  setSize(this.view.width(), this.view.
  height()); this.setImage(this.view.
  css('background-image'), startImgX,
  startImgY); this.setupFrames(fps,
  frames, rewind); }, });
```

Malheureusement, il y a beaucoup de problèmes avec cette approche. Le plus gros problème est que cette approche n'est pas normative, c'est-à-dire qu'elle n'est pas standard. Par conséquent, les développeurs qui ne sont pas habitués à cette implémentation des classes peuvent avoir des difficultés à lire ou à écrire un tel code, contrairement à ce qu'ils peuvent avoir l'habitude de rencontrer. De plus, l'implémentation exacte est cachée. Pour la connaître, le développeur doit regarder la définition originale de la classe et la façon dont elle est utilisée.

Avec TypeScript, il existe une façon unifiée de créer des classes. En outre, l'implémentation est alignée sur ECMAScript 6. Par conséquent, nous obtenons une façon portable, lisible, extensible, facile à

utiliser et standard. Si nous revenons sur l'exemple initial, nous pouvons le transformer ainsi :

```
1 class Gauge extends Base { constructor(
  id: string, startImgX: number,
  startImgY: number, fps: number,
  frames: number, rewind: boolean) {
  super(0, 0); this.view = $('#' + id)
  ; this.setSize(this.view.width(),
  this.view.height()); this.setImage(
  this.view.css('background-image'),
  startImgX, startImgY); this.
  setupFrames(fps, frames, rewind); }
  };
```

C'est relativement similaire et cela se comporte de façon à peu près identique. Cependant, le remplacement de l'implémentation précédente par la variante TypeScript doit être réalisé en une seule itération. Pourquoi? Si nous changeons la classe de base (appelé simplement Base), nous devons changer toutes les classes dérivées (les classes TypeScript doivent hériter d'autres classes TypeScript).

D'autre part, si nous changeons l'une des classes dérivées nous ne pouvons plus utiliser la classe de base. Cela étant dit, seules les classes, qui sont complètement découplées d'une hiérarchie de classe, peuvent être transformées en une seule itération. Dans le cas contraire, nous devons mettre à jour l'ensemble de la hiérarchie de classe en une seule fois.

Le mot-clé extends a une signification différente de celle pour les interfaces. Une interface étend d'autres définitions (interface ou la partie interface d'une classe) par un ensemble de nouvelles définitions. Une classe étend une autre classe en appliquant son prototype à la classe étendue. En outre, d'autres possibilités sont offertes, comme la possibilité d'accéder aux méthodes de la classe parente par l'intermédiaire de super.

La classe la plus importante est la racine de la hiérarchie de classes, appelée Base. Elle contient pas mal de fonctionnalités.

```
1 class Base implements Point, Size {
  frameCount: number; x: number; y:
  number; image: Picture; width:
  number; height: number; currentFrame
  : number; frameID: string;
  rewindFrames: boolean; frameTick:
  number; frames: number; view: JQuery
  ; constructor(x: number, y: number)
  { this.setPosition(x || 0, y || 0);
  this.clearFrames(); this.frameCount
  = 0; } setPosition(x: number, y:
  number) { this.x = x; this.y = y; }
  getPosition(): Point { return { x :
  this.x, y : this.y }; } setImage(img
  : string, x: number, y: number) {
  this.image = { path: img, x : x, y
  : y }; } setSize(width, height) {
  this.width = width; this.height =
  height; } getSize(): Size { return {
  width: this.width, height: this.
  height }; } setupFrames(fps: number,
  frames: number, rewind: boolean, id
  ?: string) { if (id) { if (this.
```

```

frameID === id) return true; this.
frameID = id; } this.currentFrame =
0; this.frameTick = frames ? (1000 /
fps / setup.interval) : 0; this.
frames = frames; this.rewindFrames =
rewind; return false; } clearFrames
() { this.frameID = undefined; this.
frames = 0; this.currentFrame = 0;
this.frameTick = 0; } playFrame() {
if (this.frameTick && this.view) {
this.frameCount++; if (this.
frameCount >= this.frameTick) { this.
frameCount = 0; if (this.
currentFrame === this.frames) this.
currentFrame = 0; var $el = this.
view; $el.css('background-position',
'-'+ (this.image.x + this.width *
((this.rewindFrames ? this.frames -
1 : 0) - this.currentFrame)) + 'px -
'+ this.image.y + 'px'); this.
currentFrame++; } } } };

```

Le mot-clé `implements` est similaire à l'implémentation des interfaces en C#. Pour l'essentiel, il s'agit d'établir un contrat où nous nous engageons à fournir dans notre classe les fonctionnalités prévues dans les interfaces mentionnées. Bien que nous ne puissions hériter que d'une seule classe (via `extends`), nous pouvons implémenter autant d'interfaces que nous le voulons. Dans l'exemple précédent, nous choisissons de ne pas hériter d'une classe, mais d'implémenter deux interfaces.

Ensuite, nous définissons les champs qui seront disponibles sur les objets de la classe en question. L'ordre n'a pas d'importance, mais il est nécessaire de les définir au préalable (et le plus important : dans un lieu unique). La méthode `constructor` est une méthode spéciale qui a la même signification que la fonction personnalisée `init` dans le code original en JavaScript. Il s'agit du constructeur de la classe comme son nom l'indique. Le constructeur de la classe parente peut être appelé à tout moment via `super()`.

TypeScript fournit également des modificateurs d'accès. Ils ne sont pas inclus dans la norme ECMAScript 6. C'est pourquoi je n'aime pas les utiliser. Néanmoins, nous pourrions rendre certains champs privés (mais n'oubliez pas : seulement du point de vue du compilateur, pas dans le code JavaScript lui-même) et donc restreindre l'accès à ces variables.

Une utilisation intéressante de ces modificateurs d'accès est possible en les combinant avec le constructeur lui-même :

```

1 class Base implements Point, Size {
    frameCount: number; // no x and y
    image: Picture; width: number;
    height: number; currentFrame: number;
    frameID: string; rewindFrames:
    boolean; frameTick: number; frames:
    number; view: JQuery; constructor(
    public x: number, public y: number)
    { this.clearFrames(); this.

```

```

frameCount = 0; } /* ... */ }

```

En précisant que les arguments sont publics, nous pouvons omettre la définition (et l'initialisation) de `x` et `y` dans la classe. TypeScript s'en chargera automatiquement.

4.6 Fonctions anonymes fléchées

Qui se rappelle comment créer des fonctions anonymes en C# avant l'introduction des lambda-expressions? Peu de développeurs C# en tout cas. Et la raison est simple : les lambda-expressions apportent expressivité et lisibilité. En JavaScript, tout évolue autour du concept de fonctions anonymes. Personnellement, je n'utilise que des expressions fonctionnelles (fonctions anonymes) au lieu de déclarations de fonctions (fonctions nommées). Cela rend les choses beaucoup plus évidentes, plus souples et apporte une certaine consistance au code. Je parlerai même de cohérence.

Néanmoins, il y a des petits passages où écrire une chose comme celle-ci n'est pas terrible :

```

1 var me = this; me.loop = setInterval(
    function() { me.tick(); }, setup.
    interval);

```

Pourquoi ce gaspillage? Quatre lignes pour rien. La première ligne est nécessaire, puisque la fonction `callback` de `setInterval` est invoquée dans le contexte `window`. Par conséquent, nous devons sauvegarder le `this` original, afin d'accéder/retrouver l'objet. Cette fermeture fonctionne. Maintenant que nous avons stocké `this` dans `me`, nous pouvons déjà profiter de ce raccourci (c'est déjà ça). Enfin, nous devons réinjecter cette fonction `callback` dans une autre fonction. Absurde? Utilisons la fonction anonyme fléchée!

```

1 this.loop = setInterval(() => this.tick
    (), setup.interval);

```

À présent, nous avons un agréable *one-liner*. Nous avons économisé une ligne en préservant `this` au sein de la fonction anonyme fléchée (appelons-la lambda-expression). Deux autres lignes qui étaient consacrées à la syntaxe de la fonction sont maintenant redondantes à cause de la lambda-expression. À mon avis, c'est non seulement plus lisible, mais également plus compréhensible.

Sous le capot, bien sûr, TypeScript génère la même chose que le code JavaScript qui précède. Mais ne nous en soucions pas. Tout comme nous ne nous soucions pas du MSIL généré par un compilateur C#, ou du code assembleur généré par un compilateur C. Nous nous soucions seulement du code source (original) comme étant beaucoup plus lisible et flexible. Si nous sommes incertains par rapport à `this`, nous devrions utiliser une fonction anonyme fléchée.

5 Étendre le projet

TypeScript compile en du JavaScript (lisible). Il aboutit à de l'ECMAScript 3 ou 5 selon la cible choisie.



Maintenant que nous avons grossièrement typé toute notre solution, nous pourrions même aller plus loin et utiliser certaines fonctionnalités de TypeScript pour rendre le code plus agréable, plus facile à faire évoluer et à utiliser. Nous allons voir que TypeScript propose des concepts intéressants, qui nous permettent de découpler entièrement notre application et la rendre accessible, non seulement dans le navigateur, mais aussi sur d'autres plates-formes telles que node.js (et donc le terminal).

5.1 Valeurs par défaut et paramètres optionnels

Nous pourrions nous satisfaire de ce que nous avons vu à ce stade, mais pourquoi en rester là ? Faisons appel aux valeurs par défaut pour certains paramètres afin de les rendre facultatifs.

Par exemple, le code TypeScript suivant sera converti...

```
1 var f = function(a: number = 0) { } f();
```

... en JavaScript comme ceci :

```
1 var f = function (a) { if (a === void 0)
  { a = 0; } }; f();
```

De cette façon, ces valeurs par défaut sont toujours définies dynamiquement, contrairement aux valeurs par défaut en C#, qui sont définies statiquement. Ceci permet de réduire le code de façon importante puisque nous pouvons désormais faire l'impasse sur pratiquement tous les contrôles de valeur par défaut et laisser TypeScript faire le travail. À noter que le `void 0` est une version sécurisée de `undefined`.

À titre d'exemple, considérons le code suivant :

```
1 constructor(x: number, y: number) { this
  .setPosition(x || 0, y || 0); // ...
}
```

Pourquoi devrions-nous nous assurer que les valeurs `x` et `y` sont définies ? Nous pouvons placer directement cette contrainte sur la fonction constructeur. Voyons à quoi ressemblerait le code modifié :

```
1 constructor(x: number = 0, y: number = 0
  ) { this.setPosition(x, y); // ... }
```

Il existe d'autres exemples. Ce qui suit montre une fonction après avoir été remaniée :

```
1 setImage(img: string, x: number = 0, y:
  number = 0) { this.view.css({
  backgroundImage : img ? c2u(img) : '
  none', backgroundColor : '-' + x
  + 'px -' + y + 'px', }); super.
  setImage(img, x, y); }
```

Encore une fois, ceci rend le code plus facile à lire. Sans cela, la propriété `backgroundPosition` aurait dû être initialisée en gérant la valeur par défaut, ce qui n'est pas très élégant.

Avoir des valeurs par défaut est évidemment pratique, mais nous pourrions aussi rencontrer une situation où nous aurions juste besoin d'omettre un argument en toute sécurité sans avoir à spécifier une valeur par défaut. Dans ce cas, à l'instar de JavaScript, nous devons toujours vérifier si un paramètre a été fourni. Cependant, un appelant peut omettre l'argument facultatif sans générer d'erreur.

L'astuce est de mettre un point d'interrogation derrière le paramètre. Regardons un exemple :

```
1 setupFrames(fps: number, frames: number,
  rewind: boolean, id?: string) { if
  (id) { if (this.frameID === id)
  return true; this.frameID = id; } //
  ... return false; }
```

Évidemment, l'appel de la méthode sans spécifier le paramètre `id` est autorisé. Par conséquent, nous devons vérifier si celui-ci existe. Cela se fait dans la première ligne du corps de la méthode. Ce garde-fou sécurise l'utilisation du paramètre optionnel, même si TypeScript nous permet de l'utiliser comme bon nous semble. Néanmoins, nous devons être prudents. TypeScript ne détecte pas toutes les erreurs ; il reste de notre responsabilité de veiller à ce que le code soit fonctionnel pour chaque chemin possible.

5.2 Surcharges

JavaScript, par nature, ne connaît pas la surcharge de fonction. La raison est assez simple : nommer une fonction ne génère qu'une variable locale et ajouter une fonction à un objet insère une entrée dans son dictionnaire. Ces deux façons ne permettent que des identificateurs uniques. Si ce n'était pas le cas, nous pourrions avoir deux variables ou deux propriétés avec le même nom. Bien sûr, il existe un moyen facile de contourner cela. Nous pouvons créer une fonction parente qui appelle des sous-fonctions, selon le nombre et le type des arguments.

Néanmoins, autant il est facile de déterminer le nombre d'arguments, autant obtenir leur type est difficile. Du moins avec TypeScript. TypeScript ne

considère les types que lors de la compilation. Ensuite, il se débarrasse de l'ensemble du système de typage. Cela signifie qu'aucune vérification de type n'est possible lors de l'exécution ; du moins pas au-delà des vérifications élémentaires de type en JavaScript.

OK, alors pourquoi consacrer un paragraphe à ce sujet si TypeScript ne peut pas nous aider ? Eh bien, parce qu'évidemment les surcharges à la compilation sont encore possibles et même nécessaires. Beaucoup de bibliothèques JavaScript contiennent des fonctions qui se comportent d'une façon ou d'une autre selon les arguments. Par exemple, jQuery offre généralement deux ou plusieurs variantes pour ses fonctions. L'une consiste à lire, l'autre à écrire une certaine propriété. Lorsque nous surchargeons des méthodes en TypeScript, nous n'avons qu'une seule implémentation avec plusieurs signatures.

En général, certains essaient d'éviter de telles constructions ambiguës, c'est pourquoi il n'y a pas de telles méthodes dans le code original du jeu. Nous n'allons pas en ajouter dès à présent, mais voyons comment nous pourrions les écrire :

```
1 interface MathX { abs: { (v: number[]): number; (n: number): number; } }
```

L'implémentation pourrait se présenter ainsi :

```
1 var obj: MathX = { abs: function(a) {
  var sum = 0; if (typeof(a) === 'number') sum = a * a; else if (Array.isArray(a)) a.forEach(v => sum += v * v); return Math.sqrt(sum); } };
```

L'avantage d'indiquer à TypeScript l'existence de différentes versions d'appel réside dans les possibilités accrues pour l'outillage. Les EDI comme Visual Studio ou des éditeurs de texte comme Bracket peuvent afficher toutes les surcharges, y compris leurs descriptions. Les appels usuels étant restreints aux surcharges fournies, cela garantit une certaine sécurité.

5.3 Types génériques

Les types génériques peuvent également être utiles pour faire face à diverses situations concernant les types. Ils fonctionnent un peu différemment qu'en C#, car ils ne sont évalués qu'au moment de la compilation. En outre, ils ne changent en rien la représentation du code au moment de l'exécution. Il n'y a aucun modèle, métaprogrammation ou quoi que ce soit ici. Les types génériques ne sont qu'une autre façon d'assurer la sécurité du typage sans que cela soit trop verbeux.

Prenons la fonction suivante :

```
1 function identity(x) { return x; }
```

Ici, l'argument x est de type any. Par conséquent, la fonction retournera quelque chose du type any. Cela peut ne pas sembler un problème, mais considérons les appels de fonction suivants.

```
1 var num = identity(5); var str =
  identity('Hello'); var obj =
  identity({ a : 3, b : 9 });
```

Quel est le type de num, str et obj ? Ils ont un nom évident, mais du point de vue du compilateur TypeScript, ils sont tous de type any.

C'est là que les types génériques viennent à la rescousse. Nous pouvons préciser au compilateur que le type de la valeur de retour de la fonction est le type du paramètre.

```
1 function identity<T>(x: T): T { return x
  ; }
```

Dans le code ci-dessus, nous retournons simplement le même type que celui en entrée de la fonction. Plusieurs scénarios existent (y compris retourner un type déterminé à partir du contexte), mais retourner le type d'un des arguments est probablement le plus courant.

Le code actuel du jeu ne contient pas de types génériques. La raison est simple : le code se focalise surtout sur les changements d'état et non sur l'évaluation des données en entrée. Ce qui fait que les traitements se font principalement à l'aide de procédures et non de fonctions. Si nous voulons utiliser des fonctions avec des paramètres pouvant être de type variable, des classes dépendant d'un type ou de constructions similaires, les types génériques sont certainement utiles. Néanmoins, jusqu'à présent, nous avons pu nous débrouiller sans eux.

5.4 Modules

La touche finale consiste à découpler notre application. Au lieu de référencer tous les fichiers, nous allons utiliser un chargeur de modules (par exemple AMD/RequireJS côté navigateur ou pour CommonJS côté serveur) et charger les différents scripts à la demande. Il existe de nombreux avantages à cette approche. Le code est beaucoup plus facile à tester, déboguer et permet en principe d'éviter les problèmes d'ordonnancement, puisque les modules sont toujours chargés après que leurs dépendances sont disponibles.

TypeScript propose une abstraction intéressante concernant la modularité dans son ensemble, car il fournit deux mots-clés (import et export) qui seront transformés en fonction du système de module souhaité. Cela signifie qu'une seule base de code peut être compilée pour se conformer à la fois à du code AMD ainsi qu'à du code CommonJS. Nul besoin de tour de magie.

Par exemple, le fichier *constants.ts* ne sera pas référencé. Au lieu de cela, le fichier va exporter son contenu sous la forme d'un module. Cela peut se faire ainsi :

```
1 export var audiopath = 'Content/audio/';
  export var basepath = 'Content/';
  export enum Direction { none = 0,
```

```
left = 1, up = 2, right = 3, down =
4, }; /* ... */
```

Comment pouvons-nous utiliser cela ? Au lieu d'avoir un référencement, nous utilisons la méthode `require()`. Pour indiquer que nous souhaitons utiliser le module directement, nous n'employons pas `var`, mais `import`. Notez que nous pouvons omettre l'extension `.ts`. C'est logique puisque le fichier aura le même nom ultérieurement, mais une extension différente.

```
1 import constants = require('./constants')
  );
```

La différence entre `var` et `import` est très importante. Considérons les lignes suivantes :

```
1 import Direction = constants.Direction;
  import MarioState = constants.
  MarioState; import SizeState =
  constants.SizeState; import
  GroundBlocking = constants.
  GroundBlocking; import CollisionType
  = constants.CollisionType; import
  DeathMode = constants.DeathMode;
  import MushroomMode = constants.
  MushroomMode;
```

Si nous avons utilisé `var`, alors nous aurions fait appel à la représentation JavaScript sous forme de propriété. La concrétisation en JavaScript de `Direction` étant seulement un objet. Or, l'abstraction en TypeScript, quant à elle, est un type qui peut être transformé sous la forme d'un objet. Bien que souvent cela ne fasse pas de différence, avec des types tels que les interfaces, les classes ou les énumérations, nous devrions privilégier `import` à `var`. Autrement nous pouvons utiliser simplement `var` pour le renommage :

```
1 var setup = constants.setup; var images
  = constants.images;
```

Est-ce tout ? Eh bien, il y a beaucoup à dire sur les modules, mais j'essaie ici d'être concis. Tout d'abord, nous pouvons utiliser ces modules pour interfacier ces fichiers. Par exemple, l'interface publique pour `main.ts` est donnée par le code suivant :

```
1 export function run(levelData:
  LevelFormat, controls: Keys, sounds
  ? : SoundManager) { var level = new
  Level('world', controls); level.load
  (levelData); if (sounds) level.
  setSounds(sounds); level.start(); };
```

Tous les modules sont ensuite rassemblés dans un fichier tel que `game.ts`. Nous chargeons toutes les dépendances puis lançons le jeu. Alors que la plupart des modules sont un regroupement d'objets, un module peut aussi être un simple objet.

```
1 import constants = require('./constants')
  ); import game = require('./main');
  import levels = require('./
  testlevels'); import controls =
  require('./keys'); import
  HtmlAudioManager = require('./
```

```
HtmlAudioManager'); $(document).
  ready(function() { var sounds = new
  HtmlAudioManager(constants.audiopath
  ); game.run(levels[0], controls,
  sounds); });
```

Le module de control est un exemple d'un module ne contenant qu'un seul objet. Cela est réalisé de la façon suivante :

```
1 export = keys;
```

Ceci affecte l'objet `export` à l'objet `keys`.

Voyons ce que nous avons désormais. En raison de la nature modulaire de notre code, nous avons inclus certains nouveaux fichiers.



Cela ajoute une autre dépendance vers *RequireJS*, mais globalement, notre code est plus robuste et plus facile à enrichir qu'auparavant. En outre, toutes les dépendances sont toujours explicitées, ce qui élimine la possibilité de dépendances inconnues. Le système de chargement de modules combiné avec l'*IntelliSense* améliore les capacités de refactorisation, et le typage fort accroît la fiabilité de l'ensemble du projet.

Bien sûr, chaque projet peut ne pas être aussi facile à remanier. Le projet était petit et a pu se baser sur du code solide, et qui n'a pas eu le temps de devenir obsolète.

En dernière étape, nous allons morceler le gros fichier `main.ts`, afin de créer de petits fichiers découplés, ne dépendant que de quelques paramètres. Ces paramètres seront injectés au début. Cependant, une telle étape ne concerne pas tout le monde. Pour certains projets, cela pourrait rendre les choses plus confuses au lieu de les rendre plus claires.

Quoi qu'il en soit, pour la classe `Matter` nous aurions le code suivant :

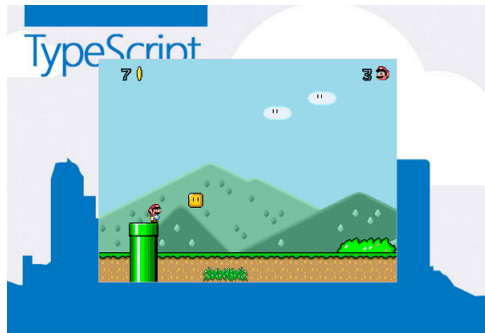
```
1 // <reference path="def/jquery.d.ts"/>
  import Base = require('./Base');
  import Level = require('./Level');
  import constants = require('./
  constants'); class Matter extends
  Base { blocking: constants.
  GroundBlocking; level: Level;
  constructor(x: number, y: number,
  blocking: constants.GroundBlocking,
  level: Level) { this.blocking =
  blocking; this.view = $('<div />').
  addClass('matter').appendTo(level.
  world); this.level = level; super(x,
  y); this.setSize(32, 32); this.
  addToGrid(level); } addToGrid(level)
  { level.obstacles[this.x / 32][this
  .level.getGridHeight() - 1 - this.y
  / 32] = this; } setImage(img: string
  , x: number = 0, y: number = 0) {
```

```
this.view.css({ backgroundImage :
img ? img.url() : 'none',
backgroundPosition : '-' + x + 'px -'
+ y + 'px', }); super.setImage(img
, x, y); } setPosition(x: number, y:
number) { this.view.css({ left: x,
bottom: y }); super.setPosition(x, y
```

```
); } }; export = Matter;
```

Cette technique devrait affiner les dépendances. En outre, la base de code gagnera en accessibilité. Néanmoins, il dépend du projet et de l'état initial du code de savoir si un tel affinement est réellement souhaitable ou simplement cosmétique.

6 Utilisation du code



Le code fonctionnel est disponible en ligne sur GitHub. Le dépôt peut être atteint via [lien 10](#). Le dépôt en lui-même contient quelques informations sur TypeScript. En outre, le système de build Gulp a été utilisé. Le dépôt contient également un guide concis sur l'installation et l'utilisation, ce qui devrait donner à chacun un point de départ pour celui qui n'a pas connaissance de Gulp ou de TypeScript.

Puisque l'origine du code réside dans l'article sur Mario5 ([lien 11](#)), je suggère également à tous ceux qui ne l'auraient pas lu d'y jeter un œil. L'article est disponible sur CodeProject. Il y a également un article sur CodeProject concernant une extension du

code source d'origine : [lien 12](#). Cette extension est un éditeur de niveau, qui met en valeur le fait que la conception du jeu Mario5 a été plutôt bonne puisque la plupart des parties de l'interface utilisateur ont pu facilement être réutilisées pour créer l'éditeur. À noter que l'article traite également d'un jeu de plateforme sociale qui combine le jeu et l'éditeur en une seule page Web, et qui peut être utilisé pour sauvegarder et partager des niveaux personnalisés.

Télécharger la démo - 9.63 Mo : [lien 13](#)

Télécharger le code source - 75 Ko : [lien 14](#).

Dépôt GitHub : [lien 15](#)

Retrouvez l'article de **Florian Rapp** traduit par **Yahiko** en ligne : [lien 16](#)

AngularJS le MVC côté client

Ce tutoriel présente le développement d'applications Web avec AngularJS. L'objectif de cet article est principalement de montrer comment concevoir une application Modèle - Vue - Contrôleur (MVC) à l'aide de ce Framework.

1 introduction

Le but de ce tutoriel est de présenter ce qu'offre AngularJS pour créer des applications basées sur le paradigme MVC côté client. Je ne présenterai pas AngularJS qui est en train de s'imposer comme le framework JavaScript de référence côté client.

Je ne présenterai pas non plus le paradigme MVC qui sépare le Modèle (les données) de la Vue (l'IHM). Le Contrôleur, quant à lui, assure la logique de contrôle et la gestion des événements. La connais-

sance de ce pattern est néanmoins un prérequis pour comprendre ce tutoriel.

On implémentera l'exemple basique d'une liste de clients que l'on pourra afficher et enrichir avec de nouveaux clients. Cet exemple nous permettra d'illustrer deux points essentiels d'AngularJS : le Databinding et l'injection de dépendances.

AngularJS étend le HTML5 pour le rendre dynamique en développant ses propres balises. Mais pour

autant si l'on veut concevoir une véritable application MVC, il ne faut pas concevoir une vue pour

la rendre dynamique, mais penser l'application en couches.

2 Installation

Le framework peut être téléchargé sur le site : [lien 17](#).

Le framework réside entièrement dans le fichier trivialement dénommé « *angular.js* ».

L'application d'illustration sera une « Single Page Application » que l'on pourra écrire à l'aide de n'importe quel éditeur de texte. Pour ma part, j'ai utilisé Sublime Text.

L'exemple que nous suivrons sera de type Standalone pour des raisons pédagogiques. Mais nous aurions très bien pu héberger les données dans une base de données distante ou avoir recours à un Web Service. D'ailleurs, si cet article ne se concentre que sur le Databinding et l'injection de dépendances, et se

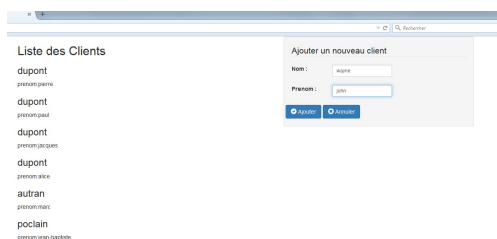
veut donc résolument simple (tout s'exécute dans un navigateur), la consommation de Web Services en AngularJS fera l'objet d'articles futurs.

Il est également à noter que dans le cas d'une application en production, on ne téléchargera pas le fichier *angular.js*, mais on choisira plutôt d'indiquer son chemin sur le site AJAX de Google et sous sa forme minimisée (sans espace ni saut de ligne) autant pour des raisons de performances que pour disposer de la dernière version du framework. Cette ligne de code sera la suivante :

```
1 <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.16/angular.min.js"></script>
```

3 Le cahier des charges

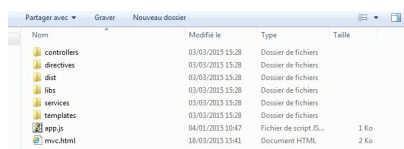
On créera une application basique permettant dans une page HTML5 (la vue) d'afficher la liste des clients déjà présents dans le modèle et d'y ajouter de nouveaux clients. Comme ci-dessous :



Le seul objectif de cet exemple d'illustration est de manipuler des clients qui constituent le « modèle » et de les faire apparaître dans la « vue » grâce à la magie du « contrôleur ». On verra également comment ce même principe permet de créer des clients dans la vue et les mettre en persistance dans le modèle.

4 L'architecture AngularJS

Voici l'architecture standard conseillée pour une application MVC depuis la racine de l'application :



On a l'habitude, pour développer avec AngularJS, de ranger les différents fichiers dans des répertoires conventionnels.

Le fichier *mvc.html* sera le fichier HTML5 que l'utilisateur connaîtra comme URL unique.

le fichier *app.js* est le fichier qui définit les modules AngularJS de notre application. Ces modules peuvent être vus comme des containers pour les différentes parties de l'application (au sens fonctionnel). On pourrait imaginer un module com-

mande, facturation ou logistique. Ici nous n'aurons qu'un seul module, donc le fichier *app.js* ne contiendra que l'unique ligne de code suivante : `angular.module("app", []);` « *app* » sera le nom de notre module et on met entre `[]` les dépendances (éventuelles) du module.

Le modèle sera dans le répertoire *services*.

Le répertoire *directives* sera utilisé pour créer de nouvelles balises (directives) à utiliser dans la vue (*mvc.html*).

Le répertoire *templates* sera utilisé pour stocker les consignes d'affichage (esthétique) des *directives*.

Le répertoire *contrôleurs* abritera les contrôleurs permettant de lier la vue et le modèle.

Le répertoire *libs* contiendra le fichier *angular.js* et le répertoire *dist* hébergera la bibliothèque *bootstrap* de Twitter que j'utilise pour la présentation HTML5.

5 Le modèle

Le modèle se trouve dans le fichier `/services/-clientsFactory.js` :

```

1 angular.module("app").factory("
  clientsFactory", function ()
2 {
3   var clients = [
4     {id:1, nom: 'dupont',
      prenom: 'pierre'},
5     {id:2, nom: 'dupont',
      prenom: 'paul'},
6     {id:3, nom: 'dupont',
      prenom: 'jacques'},
7     {id:4, nom: 'dupont',
      prenom: 'alice'}
8   ];
9   var getClients = function()
10  {
11    return clients;
12  };
13  var addClient = function(client)
14  {
15    var client = prepareClient(
16      client);
17    clients.push({id:client.id, nom:
18      client.nom, prenom:client.
19      prenom});
20  };
21  function prepareClient(client)
22  {
23    client.id = clients.length + 1;
24    return client;
25  }
26 }
27 );

```

```

22   }
23   return {
24     getClients: getClients,
25     addClient: addClient
26   };
27 });

```

Nous créerons en réalité un service (*clientsFactory*) que nous pourrions appeler partout dans notre application.

Ce service est créé grâce à la fonction *factory()* du framework qui permet réellement l'injection de dépendances.

Cette fonction *factory()* prend deux paramètres en entrée :

- le nom du service *clientsFactory* ;
- et une fonction dite de *callback*.

Depuis l'avènement d'AJAX et de son célèbre *XmlHttpRequest*, les fonctions de callback sont la clef de voûte des frameworks JavaScript côté client (comme AngularJS) ou côté serveur (comme NodeJS). Et tout ce que fait notre service se passe dans cette fonction.

La syntaxe de ce code source peut surprendre les habitués de la programmation impérative, car il s'agit là de programmation déclarative. Sans pousser trop loin la théorie, ce qu'il faut retenir de ce service, c'est que dans le *return* de sa fonction de *callback* il expose deux fonctions que l'on pourra appeler :

- *getClients* qui liste tous les clients de la liste ;
- *addClient* qui ajoute un client à la liste.

6 Le contrôleur

Notre contrôleur se trouve dans le fichier *controller/mainController* :

```

1 angular.module("app").controller("
  mainController", function ($scope,
  clientsFactory)
2 {
3   $scope.clients = clientsFactory.
  getClients();
4
5   $scope.addClient = function(client)
6   {
7     clientsFactory.addClient(client)
8     ;
9     $scope.newClient.nom='';
10    $scope.newClient.prenom='';
11  }
12 });

```

Nous créons un contrôleur (*mainController*) et là encore tout se passe dans la fonction de *callback*. Cette fonction prend deux paramètres d'entrée :

- *\$scope* : défini par la communauté AngularJS comme un vecteur vers le modèle et comme un contexte d'exécution pour les expressions dans la vue ;
- et le service défini au chapitre précédent.

Dans le corps de la fonction, le contrôleur définit dans l'objet *\$scope* la liste des clients et une méthode permettant de rajouter un client à cette liste. *\$scope* étant la glu entre la vue et le contrôleur, la liste des clients et la méthode pour en ajouter sont disponibles dans la vue.

7 Les directives

Nous allons créer une directive dont nous nous servirons dans la vue. Comme nous manipulons des

clients, il serait intéressant de créer une nouvelle ba-

lise `<client>` `</client>`.

Nous créerons cette directive dans le fichier *directives/client.js* :

```

1 angular.module("app").directive("client"
2   , function ()
3   {
4     return{
5       restrict: 'E',
6       templateUrl: 'templates/client.
7         html'
8     }
9   });

```

On remarque que la création de la directive *client* est accompagnée de sa fonction traditionnelle. Dans

cette fonction, on précise que cette directive ne s'appliquera qu'aux éléments (*restrict : 'E'*) et où se trouve le modèle HTML qu'on souhaite appliquer à cette directive.

Jetons maintenant un coup d'œil à ce modèle *templates/client.html* :

```

1 <h3>{{client.nom}}</h3>
2 prenom:{{client.prenom}}

```

Cette notation entre deux accolades permet d'évaluer dans la vue des expressions AngularJS. Ici en particulier, on affichera le nom et le prénom du client.

8 La vue

La vue est constituée pour notre application d'une simple page HTML5 *mvc.html* :

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <link href="dist/css/bootstrap.css" rel="stylesheet">
6     <title>Comprendre Angularjs</title>
7     <script src="libs/angular.js"></script>
8     <script src="app.js"></script>
9     <script src="services/clientsFactory.js"></script>
10    <script src="controllers/mainController.js"></script>
11    <script src="directives/client.js"></script>
12  </head>
13  <body>
14    <section class="container" ng-app="app" ng-controller="mainController">
15      <form name="newClientForm" class="well form-inline pull-right col-lg-5">
16        <legend>Ajouter un nouveau client</legend>
17        <fieldset class="row">
18          <label for="nom" class="col-lg-3">Nom :</label>
19          <input id="nom" type="text" style="width:150px" class="input-sm form-control" ng-model="newClient.nom">
20        </fieldset>
21        <h1></h1>
22        <fieldset class="row">
23          <label for="prenom" class="col-lg-3">Prénom :</label>
24          <input id="prenom" type="text" style="width:150px" class="input-sm form-control" ng-model="newClient.prenom">
25        </fieldset>
26        <h1></h1>
27        <fieldset class="row">
28          <button class="btn btn-primary" type="submit" ng-click="addClient(newClient)"><span class="glyphicon glyphicon-ok-sign"></span> Ajouter </button>
29          <button class="btn btn-primary" type="reset"><span class="glyphicon glyphicon-remove-sign"></span> Annuler </button>
30        </fieldset>
31      </form>
32      <h2>Liste des Clients</h2>
33      <article ng-repeat="client in clients">
34        <client />
35      </article>
36    </section>
37  </body>
38 </html>

```

Dans la section d'entête, on trouve l'importation des scripts précédents et de la bibliothèque CSS Bootstrap.

Dans le corps du HTML, on remarque que toutes les instructions AngularJS sont préfixées par *ng-*. On déclare dans la balise `<section>` le module et le contrôleur.

On remarque comme il est aisé de mettre en place un Databinding bidirectionnel entre les propriétés du client et celles du formulaire à l'aide de l'instruction *ng-model*.

De même, on fera une boucle sur la liste des clients avec l'instruction *ng-repeat* afin d'afficher les informations des clients grâce à la directive `<client>` que l'on a créée.

9 Conclusion

Cet article a permis d'exposer comment mettre en place simplement un MVC avec AngularJS. Ce n'est cependant qu'un début pour deux raisons :

- nous ne sommes pas dans cet exemple sur une architecture de type Internet. En effet, il faudrait que les données du modèle soient issues d'un serveur tiers, via un service REST qu'An-

gularJS sait parfaitement consommer à travers son objet *\$resource* ;

- nous n'utilisons qu'une petite partie de ce qu'AngularJS sait faire. Parmi les apports intéressants d'AngularJS, ne citons que les *routes*, les *promises* et la manipulation d'AJAX.

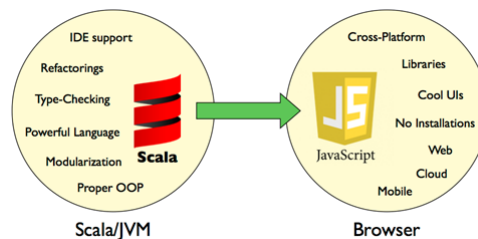
Retrouvez l'article de **Marc Autran** en ligne : [lien 18](#)



Java

Les dernières news Java

Le langage Scala s'ouvre au Web - Scala.js permet de compiler du code Scala en JavaScript



Les concepteurs du langage Scala ont publié un nouveau compilateur appelé scala.js. Avec cette nouvelle extension, il devient possible pour les développeurs de créer, totalement en langage Scala, des applications Web.

Les principales caractéristiques de Scala.js sont :

- Support natif de tous les modules Scala existants ;
- Très bonne interopérabilité avec du code JavaScript. Il est par exemple possible d'utiliser JQuery ou HTML5 à partir du code Scala.js ;
- Possibilité de générer des scripts JavaScript Source Maps avec un débogage en douceur : [lien 19](#) ;
- Bonne intégration avec l'outil de Google, Closure Compiler ([lien 20](#)) ;
- Génération du JavaScript optimisé ;
- Peut être facilement utilisé avec votre EDI favori pour Scala.

Scala.js introduit certaines bibliothèques spécifiques à JavaScript comme scala-js-jquery, scala-dom et scala-js-pouchdb, ainsi que le support de plusieurs frameworks, à l'instar des frameworks de test comme Utest et MiniTest, pour faciliter le développement d'applications Web robustes.

La documentation sur scala.js ([lien 21](#)) est assez fournie pour une prise en main en souplesse de ce

compilateur. En plus, l'équipe de développement a mis en place un groupe de discussion ([lien 22](#)) et un salon de chat ([lien 23](#)) pour vos questions et vos retours d'expérience.

Il convient aussi de remarquer qu'avec la portabilité que lui apporte son exécution dans la JVM (Java Virtual Machine), et sa simplicité dans la programmation orientée objet, Scala consolide sa position face au géant Java, jugé par certains développeurs, trop exigeant par sa verbosité. D'ailleurs, la transition de Java à Scala, est facilitée par la capacité d'évoquer du code écrit en Scala à partir de programmes écrits en Java : [lien 24](#).

D'après Graham Tackley du quotidien britannique The Guardian, « Scala permet de faire plus, avec peu de code. Il apporte une bouffée d'air frais aux développeurs. »

Supporté en environnement Microsoft .Net ([lien 25](#)), Scala présente un fort potentiel d'interopérabilité qui peut aller jusqu'au développement d'applications mobiles pour les différents systèmes d'exploitation mobile : Android, Windows Phone et iOS.

Pour rappel, Scala est le produit de longues années de recherches du professeur Martin Odersky de l'École polytechnique fédérale de Lausanne. Son objectif initial était d'unifier la programmation orientée objet et la programmation fonctionnelle, avec un typage statique.

Commentez la news de **Siguillaume** en ligne : [lien 26](#)

Les derniers tutoriels et articles

Tutoriel sur les bases avec JavaFX

JavaFX est la bibliothèque graphique remplaçante de Swing et de AWT. Elle a pour avantage d'être utilisable via un langage objet et statiquement typé.

1 Introduction

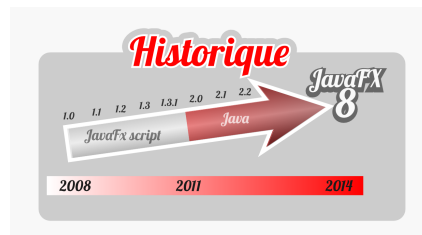
JavaFX est une bibliothèque graphique intégrée dans le **JRE** et le **JDK** de **Java**. **Oracle** la décrit comme « The Rich Client Platform », c'est-à-dire qu'elle permet de réaliser des interfaces graphiques évoluées et modernes grâce à de nombreuses fonctionnalités, telles que les animations, les effets, la 3D, l'audio, la vidéo, etc. Elle a de plus l'avantage d'être dans le langage Java, qui permet de réaliser des architectures avec des paradigmes objet, et aussi de pouvoir utiliser le typage statique.

Dans ce premier tutoriel, nous allons voir ensemble un rapide historique de la bibliothèque pour

ensuite découvrir les fondamentaux que sont les classes « **Stage** », « **Scene** », « **Application** » et le « **threading** » associé, pour finir nous verrons les « **Node** » avec un exemple d'utilisation du « **scene graphe** ». Cette présentation ne fait pas dans le bling-bling, même si **JavaFX** est doué pour cela, en préférant se focaliser sur les concepts primordiaux d'une telle bibliothèque. Bien comprendre ces basiques vous aidera à bien commencer pour ensuite pouvoir faire des interfaces de qualité et peut-être spectaculaires.

2 Historique

La première version de **JavaFX** 1.0 a été créée en 2008. Elle était bien différente de la version courante 8, car elle était utilisable via un langage script spécifique. Il fallait aussi la télécharger à part et le support des outils était faible. A contrario maintenant la maturité aidant, elle est incluse par défaut dans **Java** et directement accessible dans les **IDE**. De plus, il est inutile d'apprendre un nouveau langage, car l'**API** est comme **Swing** et **AWT** en **Java**.



Cette information peut s'avérer pertinente, car vous pouvez encore trouver sur Internet des documents ou autres exemples de ces anciennes versions obsolètes qu'il ne faut pas utiliser. Le langage **Java** a été introduit à partir de la version 2.0.

Une autre information importante est que **JavaFX** est la bibliothèque remplaçante de **Swing** qui elle-même remplaçait **AWT**. Dans les versions courantes 1.8, les trois bibliothèques sont toujours accessibles, mais seule **JavaFX** va, dans le futur, avoir de nouvelles fonctionnalités.

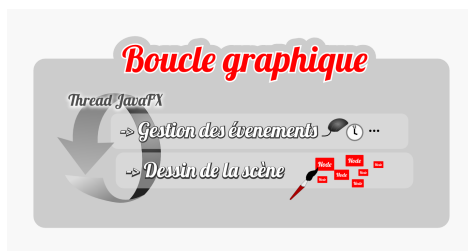
Les deux autres sont considérées comme des branches mortes. Donc si vous partez sur de nouveaux développements d'IHM en Java, je vous conseille d'utiliser **JavaFX** qui est la seule à avoir le support d'**Oracle** pour les prochaines années.

3 Premières lignes de code

Commençons par le début, c'est-à-dire comment lancer une application de type **JavaFX**. Comme classiquement en Java, le démarrage est réalisé via une classe contenant une méthode statique portant le nom « **main** ». Par contre et contrairement à **Swing** ou à **AWT**, **JavaFX** vous oblige à étendre la classe « **javafx.application.Application** ». Cela a

pour but de forcer les développeurs à utiliser **JavaFX** dans le **Thread JavaFX**. En effet comme la grande majorité des bibliothèques graphiques, l'utilisation de **JavaFX** est **monthread**, c'est-à-dire que tous les appels à l'**API** ou la création d'objets de type **JavaFX** doivent être réalisés dans ce thread. Le schéma, ci-dessous, vous indique de manière sim-

plifiée le travail du thread.



Attention si vous utilisez l'API en dehors de ce thread votre IHM risque fortement d'avoir un comportement incongru ceci agrémenté de levées d'exceptions qui peuvent être difficiles à comprendre.

Ci-dessous, vous trouverez un exemple qui n'affiche rien, mais qui permet de comprendre la notion d'application ainsi que de comprendre le threading. Maintenant que **JavaFX** est inclus dans le **JDK**, il n'y a plus de contrainte particulière pour l'utiliser. Utilisez votre IDE préféré comme **Eclipse**, **Netbeans** ou **IntelliJ**. Configurez bien l'utilisation d'un JDK supérieur à la version 1.8 et c'est tout.

```
1 import javafx.application.Application;
```

4 Bienvenue au théâtre

Écrire une bonne API est un art difficile. De plus c'est une activité réalisée par l'Homme pour l'Homme. Sachant cela, une bonne méthode de création est d'avoir recours à des analogies connues par le plus grand nombre d'entre nous. Pour leur part, les créateurs de **JavaFX** ont choisi des terminologies provenant du théâtre.

- **javafx.stage.Stage** : peut-être traduit par l'estrade. C'est le stage qui définit la taille de la fenêtre et aussi son titre. En **AWT** ou **Swing** cela correspondait à une **Frame** ou **Jframe**. C'est le lien avec le système de fenêtre des systèmes d'exploitation.
- **javafx.scene.Scene** : correspond au décor où a lieu l'action. Comme au théâtre il est possible d'avoir plusieurs décors pour une même

5 Afficher vos premières nodes

Pour afficher vos premiers objets graphiques, il suffit de créer un « Scene Graph ». Derrière ce mot se cache juste un arbre d'éléments graphiques qui permet de définir, dans un repère, où vos éléments vont se dessiner et aussi dans quel ordre. Comme tout arbre, nous retrouvons les notions classiques d'élément de type parent ou de type feuille.

Pour les utilisateurs de **AWT** ou **Swing** cela existait déjà, avec des « **Component** », « **JComponent** » et des « **Container** ». La différence vient

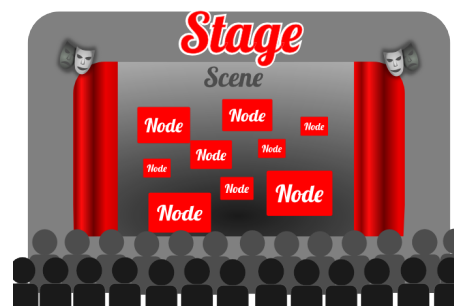
```
2 import javafx.stage.Stage;
3
4
5 public class MyFirstJfxApplication
6     extends Application{
7
8     public static void main(String[]
9         args) {
10         System.out.println( "Main method
11             inside Thread : " + Thread
12                 .currentThread().getName());
13         launch(args);
14     }
15
16     @Override
17     public void start(Stage args) throws
18         Exception {
19         System.out.println( "Start
20             method inside Thread : " +
21                 Thread.currentThread().
22                 getName());
23     }
24 }
```

Exécutez cette classe et vous devez avoir, dans la console, le texte suivant :

```
Main method inside Thread : main
Start method inside Thread : JavaFX Application Thread
```

estrade.

- **javafx.scene.Node** : ils n'ont pas poussé l'analogie, jusqu'au bout, en utilisant le terme acteur. Mais les nodes se placent dans le décor comme des acteurs.



que cette fois **JavaFX** propose des primitives graphiques telles que le rectangle, le cercle, etc., et pas uniquement des gadgets graphiques comme le bouton, les listes de choix, etc.

Il y a plusieurs types de nodes :

javafx.scene.Canvas : nous n'allons pas voir ce type de nodes, mais sachez qu'il permet d'avoir un « graphic context » comme en Java2D, cela permet de réaliser des dessins très élaborés avec une facilité pour l'optimisation. De là, vous pouvez implémenter

votre propre « Scene Graph » ;

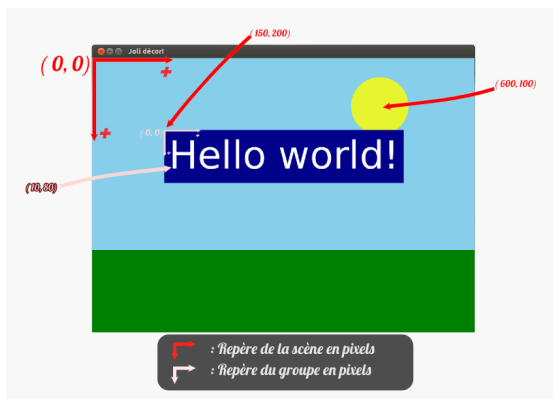
javafx.scene.ImageView : ce type de nodes sert à la manipulation et l'affichage d'images ;

javafx.scene.MediaView : celui-ci sert à afficher de la vidéo ou jouer des sons ;

javafx.scene.Parent : nous utiliserons un parent de type « Group » dans l'exemple suivant. Les parents servent à composer les éléments graphiques ;

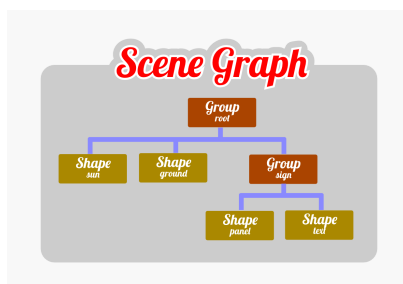
javafx.scene.Shape : pour illustrer le « Scene Graph », nous allons utiliser ce type de nodes, car ce sont les plus simples. Elles permettent d'afficher des éléments graphiques basiques tels que les rectangles, cercles, textes, etc.

Dans l'exemple de code, fourni plus bas, nous allons afficher l'application suivante composée d'une scène avec un fond bleu, contenant un soleil, un sol vert, un panneau lui-même composé de deux éléments : un panneau bleu foncé et un texte blanc.



Le système de coordonnées est de type « Device », c'est-à-dire que l'unité est le pixel, que l'origine est située en haut à gauche et que les sens positifs sont dirigés vers la **droite** et vers le **bas**. Ceci est très classique dans le monde du graphisme.

Pour réaliser cette application, une arborescence de nodes doit être créée correspondant au diagramme ci-dessous. Vous pouvez observer que pour la composition, il faut utiliser un parent de type Group. Ces enfants sont alors placés dans le repère du parent et non dans celui de la scène. Vous remarquerez aussi que l'ordre des nodes est utilisé pour définir l'ordre de dessin. C'est grâce à cela que le soleil est dessiné derrière le panneau.



Le code source est le suivant. Reprenez l'application précédente et remplacez la méthode « start » avec celle-ci :

```

1  @Override
2  public void start(Stage stage)
   throws Exception {
3      // définit la largeur et la
   hauteur de la fenêtre
4      // en pixels, le (0, 0) se situe
   en haut à gauche de la fenê
   tre
5      stage.setWidth(800);
6      stage.setHeight(600);
7      // met un titre dans la fenêtre
8      stage.setTitle("Joli décor!");
9
10     // la racine du sceneGraph est
   le root
11     Group root = new Group();
12     Scene scene = new Scene(root);
13     scene.setFill(Color.SKYBLUE);
14
15     // création du soleil
16     Circle sun = new Circle(60,
   Color.web("yellow", 0.8));
17     sun.setCenterX(600);
18     sun.setCenterY(100);
19
20     // création du sol
21     Rectangle ground = new Rectangle
   (0, 400, 800, 200);
22     ground.setFill(Color.GREEN);
23
24     // création d'un élément plus
   complexe, le panneau
25     Group sign = new Group();
26     sign.setTranslateX(150);
27     sign.setTranslateY(200);
28     // Attention les coordonnées
   sont celles du panneau, pas
   de la scène
29     Text text = new Text(10, 30, "
   Hello world!");
30     text.setFont(new Font(80));
31     text.setFill(Color.WHITE);
32     // le repère utilisé est celui
   du panneau
33     Rectangle panel = new Rectangle(
   0, -50, 500, 110);
34     panel.setFill(Color.DARKBLUE);
35     // composer l'élément plus
   complexe
36     sign.getChildren().add(panel);
37     sign.getChildren().add(text);
38
39     // ajout de tous les éléments de
   la scène
40     root.getChildren().add(sun);
41     root.getChildren().add(ground);
42     root.getChildren().add(sign);
43
44     // ajout de la scène sur l'
   estrade
45     stage.setScene(scene);
46     // ouvrir le rideau
47     stage.show();
48 }

```

6 Conclusion

Nous avons vu ensemble les concepts de **JavaFX** et l'analogie avec le théâtre. De plus nous avons regardé l'application **JavaFX** avec son threading ainsi que la composition du « Scene Graphe » par la création d'une arborescence de « Node ».

Même si ce que nous avons vu ensemble n'est pas spectaculaire, cela n'est pas moins important, car cela correspond aux bases de **JavaFX**. Ces bases sont classiques et communes à de nombreuses bibliothèques graphiques. Il est donc important de

bien les comprendre pour les professionnels du graphisme et autres amateurs éclairés.

Sur ces bases, il y a encore une grande richesse fonctionnelle à découvrir avec les effets, les animations, la 3D, les CSS, etc.

Pour aller plus loin :

- [lien 27](#)
- [lien 28](#)
- [lien 29](#)

Retrouvez l'article de **Mik Arber** en ligne : [lien 30](#)

Test d'IntelliJ IDEA v14

Que nous apporte la nouvelle version de l'éditeur IntelliJ IDEA de JetBrains ?

Dans cet article, nous allons voir les nouveautés apportées par la 14^e version d'IntelliJ IDEA de JetBrains.

1 Qu'est-ce qu'IntelliJ

Faisons en premier lieu un tour rapide de l'outil.

1.1 Introduction

Depuis janvier 2001, la société JetBrains ([lien 31](#)) édite le logiciel IntelliJ IDEA ([lien 32](#)). Il s'agit d'un **EDI** (ou **IDE** en anglais), à savoir un *Environnement de Développement Intégré (Integrated Development Environment)*, autrement dit un ensemble d'outils destinés au développement logiciel. IDEA est ainsi à mettre au même niveau - toutes proportions gardées - qu'Eclipse ([lien 33](#)) ou encore que NetBeans ([lien 34](#)).

Pour information, **IntelliJ** fait référence à la plateforme commune de JetBrains pour tous ses outils de développement, **IDEA** étant l'EDI de développement Java. Il est donc plus juste de dire « je travaille sur IDEA » que « je travaille sur IntelliJ », bien que ce soit souvent la seconde phrase qui soit la plus courante.

1.2 Versions et prix

IDEA existe en deux versions : **Community** et **Ultimate**. Pour faire simple, la version **Community**, gratuite, est avant tout destinée au développement d'applications « lourdes » Java, Scala et Android. Dès qu'il s'agit de développer des applications web, il faut se tourner vers l'édition **Ultimate**. Son prix, pour une licence personnelle, est de 191 € (hors

promotion ou prix de mise à jour), une licence commerciale est à 479 €. 50 % de réduction sont accordés aux mises à jour ainsi que pour les start-up. Les étudiants, les enseignants ou les projets open source peuvent, quant à eux, disposer de versions complètement gratuites !

Un comparatif complet des deux éditions est visible ici : [lien 35](#).

Vous pouvez télécharger l'une des deux versions sur la page dédiée : [lien 36](#). Notez au passage qu'IntelliJ IDEA est compatible Windows, Mac et Linux.

1.3 Principales fonctionnalités

Cet article n'a pas pour but de détailler toutes les fonctionnalités d'IDEA, un livre entier serait nécessaire pour accomplir cette tâche. Nous verrons toutefois les principales fonctionnalités et langages, frameworks ou outils supportés.

1.3.a Langages et frameworks supportés

La version **Community** gère nativement les langages suivants : Java, Scala, Groovy, Clojure et XML, XSD et DTD. Avec la version **Ultimate** s'ajoutent les langages dédiés au développement web, à savoir le HTML, CSS, JavaScript, CoffeeScript, ActionScript. Viennent également le support du Freemarker, de Velocity, du XSL, XPath, SQL, Ruby et JRuby, Python ou encore PHP.

Certains de ces langages nécessitent toutefois l'ajout de plugins gratuits.

Avec la version **Ultimate**, vient également le support des frameworks les plus courants pour le développement autour de l'écosystème de la JVM ou du web. On citera par exemple Spring, Play! framework, JavaEE, GWT, Hibernate, Struts, Grails, Griffon, Sass, LESS, Rails, Django, Node.js, etc.

1.3.b Gestionnaires de sources

Si vous travaillez sur CVS (réveillez-vous, nous sommes en 2015), SVN, Mercurial ou Git (voire GitHub), alors la version **Community** sera suffisante pour vous. Si vous avez Team Foundation Server, Perforce, ClearCase ou encore Visual SourceSafe, c'est vers la version **Ultimate** qu'il vous faudra vous tourner.

1.3.c Outils de construction

Les principaux outils de construction d'applications sont présents dans les deux éditions d'IDEA. On y retrouve ainsi Maven, Gradle, Ant et Gant Build Tools.

L'édition **Ultimate**, grâce à ses capacités en développement web, supporte également des outils de construction dédiés à cet univers, comme Grunt : [lien 37](#).

2 Les nouveautés de la version 14

Comme à chaque fin d'année, JetBrains publie sa nouvelle version majeure de son EDI. C'est bien entendu encore le cas en cette fin d'année 2014, qui voit la version 14 d'IDEA sortir. Passons en revue les principales nouveautés de l'outil.

2.1 Décompilateur intégré

Je me souviens que lorsque j'utilisais Eclipse (temps heureusement révolu!), la première chose que je faisais après avoir installé l'outil, c'était de lui ajouter un « décompilateur » (Jadclipse en l'occurrence : [lien 42](#)). Quel est le rôle d'un décompilateur? C'est tout simplement de réaliser l'opération inverse du compilateur, autrement dit transformer du code compilé, donc binaire (les fameux *.class* en Java) en code source, bien plus compréhensible par l'être humain. Ce type d'outil s'avère vite indispensable dans la mesure où l'on récupère très fréquemment des dépendances compilées, sans toujours avoir les sources qui vont avec.

La première grosse amélioration de cette nouvelle version d'IDEA est l'arrivée d'un décompilateur intégré nativement dans l'EDI. Plus besoin donc d'ins-

1.3.d Développement et autres fonctions

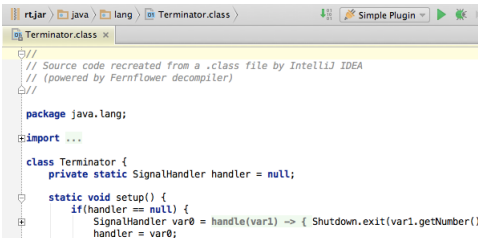
En ce qui concerne le développement à proprement parler, IDEA offre une excellente intégration des outils de tests (JUnit, TestNG, Spock ou encore Cucumber), un historique local des modifications de fichiers, une interopérabilité avec Eclipse, ou encore un gestionnaire de contextes. Ce dernier permet de travailler sur un ticket JIRA (ou n'importe quel autre gestionnaire de tickets) et d'y associer un contexte. Ainsi, lorsque l'on rouvre un ticket JIRA sur lequel on avait déjà commencé à travailler, IDEA va rouvrir les fichiers qui étaient ouverts lorsque le contexte de ce ticket avait été précédemment fermé. Si vous êtes adeptes de Mylyn sur Eclipse ([lien 38](#)), cette fonction devrait vous intéresser. Vous pouvez voir ici pour plus de détails (en anglais) : [lien 39](#).

À ce propos, IDEA s'interface sans problème avec les plus populaires des systèmes de tickets : JIRA, YouTrack, Lighthouse, GitHub, Redmine, Trac, etc.

Si vous possédez la version **Ultimate**, vous disposerez également d'un gestionnaire complet de bases de données (éditeur SQL, définition de schémas, diagrammes, etc.), d'un outil de modélisation UML, d'outils de couverture de code, du « Structural Search and Replace » (j'avais parlé de cette fonctionnalité l'an passé dans un autre article sur l'outil de JetBrains : [lien 40](#)).

L'intégralité des fonctionnalités d'IDEA est à trouver sur cette page : [lien 41](#).

taller des bibliothèques tierces ou des plugins.



```

// Source code recreated from a .class file by IntelliJ IDEA
// (powered by Fernflower decompiler)

package java.lang;

import ...

class Terminator {
    private static SignalHandler handler = null;

    static void setup() {
        if(handler == null) {
            SignalHandler var0 = handle(var1) -> { Shutdown.exit(var1.getNumber())
            handler = var0;
        }
    }
}
    
```

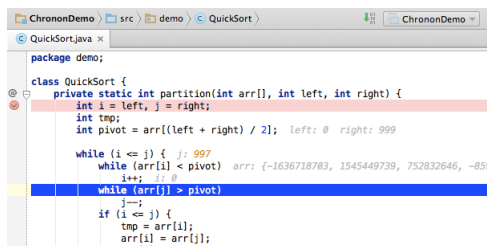
Code décompilé par IntelliJ IDEA

Cette fonctionnalité est proposée dès la version **Community**.

2.2 Débogueur amélioré

Écrire des tests est devenu aujourd'hui une étape obligatoire pour tout développement réalisé dans les règles de l'art. Cela n'empêche cependant pas d'avoir besoin, par moments, d'un débogueur. Bien entendu, IntelliJ IDEA en dispose d'un depuis déjà bien longtemps. Mais le voici encore amélioré avec cette nouvelle mouture de l'EDI. Parmi les améliorations, il en est une qui n'est pas négligeable, et qui fera sans

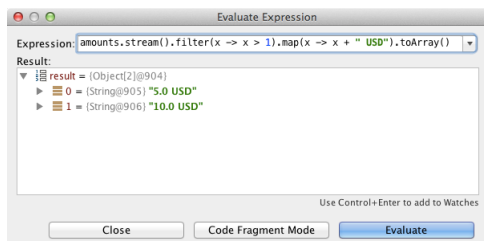
doute hurler de joie tout développeur en train de se bagarrer avec son code. En effet, IntelliJ IDEA propose « tout simplement » d'afficher les valeurs des variables directement à côté du code, comme cela est monté sur la capture de l'écran suivante :



Le débogueur en action, montrant ainsi les valeurs des variables directement dans le code

L'idée est diablement simple, et elle fera gagner un temps précieux aux développeurs. On ne peut qu'en remercier l'équipe de JetBrains!

Pour les développeurs déjà passés à Java 8, ils seront heureux d'apprendre que l'évaluateur d'expressions permet désormais de travailler avec des expressions lambdas.



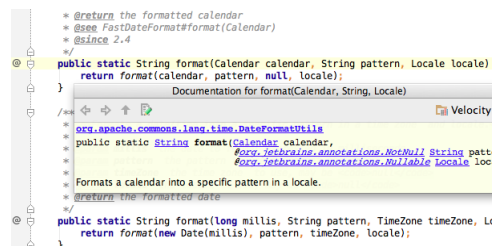
L'évaluation d'une expression lambda en Java 8

2.3 Inférence d'annotations

Les annotations ont été introduites en Java lors de la sortie de la version 5, il y a déjà dix ans ! Ces annotations sont utilisées partout non seulement pour ajouter des fonctionnalités à certaines parties du code, mais aussi pour donner plus de sens au code lui-même. C'est par exemple le cas des annotations `@Nullable` ou `@NotNull`, indiquant si une variable accepte ou non d'avoir une valeur vide (`null`). Ce genre d'indications peut être très utile pour le développeur souhaitant utiliser votre API.

Le seul hic de ces annotations, c'est qu'elles doivent être ajoutées manuellement dans le code. Eh bien plus maintenant ! En effet, IntelliJ IDEA est désormais capable, par la seule analyse du code (ou plus précisément du *byte-code*), de déterminer la pertinence de certaines annotations automatiquement, en particulier `@Nullable`, `@NotNull` ou encore `@Contract` (documentation de cette annotation : [lien 43](#)). Le code en question n'est pas modifié (ce qui implique que cette fonctionnalité marche également sur du code tiers), mais va impacter la documentation ainsi que la détection de potentiels problèmes dans votre code. Ainsi, appeler une méthode

avec un paramètre `null`, alors qu'IntelliJ IDEA aura détecté que la méthode appelante n'autorise pas ce type de valeur (quand bien même le développeur de cette méthode n'aura pas pris soin d'ajouter `@NotNull`), lèvera une erreur dans l'éditeur d'IDEA.

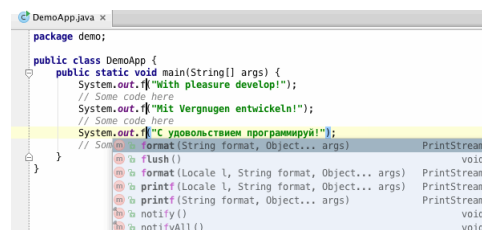


IDEA a détecté que `pattern` peut être `null`, alors que `locale` peut l'être, bien que le code ne spécifie rien de tel.

2.4 Autres améliorations

Il existe de nombreuses autres petites (ou moins petites) améliorations dans cette version d'IntelliJ IDEA. En voici quelques-unes qui méritent votre attention.

Tout d'abord, l'éditeur de code - c'est tout de même ici que nous passons le plus clair de notre temps sur un EDI - dispose de diverses nouveautés, avec une détection automatique de style de codage, une sélection multiple encore plus performante (à la Sublime Text), une meilleure gestion de la touche « Backspace », etc.



Sélection multiple au sein de l'éditeur de code.

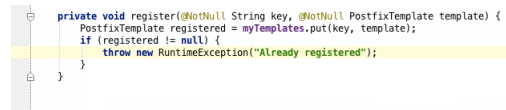
De nouveaux frameworks viennent s'ajouter à la liste - déjà longue - de ceux supportés par l'EDI. Nous pourrions citer Thymeleaf, Phonegap, Cordova ou encore Ionic. Attention toutefois, ceci n'est proposé qu'avec la version **Ultimate**. Les supports d'autres frameworks sont également améliorés. Citons en vrac : Scala, Android, Spring, GWT, Gradle, Maven ou JavaFX.

L'introduction des fichiers brouillons est aussi une bonne nouvelle. Il n'est ainsi plus nécessaire de créer physiquement un fichier pour tester quelque chose ou pour noter une idée.

L'écran des préférences a également été refait pour le rendre plus lisible. Nous noterons ainsi différentes améliorations des performances, permettant de démarrer l'outil plus rapidement que dans les versions précédentes.

Je terminerai par une fonctionnalité que j'ai découverte avec cette version 14, bien qu'elle fût introduite lors de l'édition précédente : l'édition « postfix ». De quoi s'agit-il au juste ? Imaginez que vous êtes en train de saisir un bout de code, et que vous avez oublié d'encapsuler cette ligne dans un bloc de test de non-nullité. Plutôt que de vous arrêter et d'écrire une ligne au-dessus, un bloc « *if (maVariable != null) {* », il vous suffit d'écrire à la fin (d'où le « postfix ») de la variable un « *.nonnull* », et IDEA vous corrigera (d'où le « postfix ») automatiquement votre code. C'est le genre de chose

qui vous fait adorer IntelliJ IDEA : il vous propose des fonctionnalités dont vous avez toujours rêvé sans que jamais vous n'y ayez songé ! La version 14 permet désormais de faire ce genre de chose sur du code JavaScript, alors que cela était resté limité au code Java jusqu'à présent.



La fonction de postfix permet entre autres d'ajouter un « throw » oublié.

3 Ce qui fait la force d'IntelliJ

On reproche souvent à IntelliJ IDEA d'être payant et d'avoir un prix relativement élevé. Une licence unique coûte 191 € si elle est personnelle, 479 € s'il s'agit d'une licence d'entreprise. Pour cette dernière, cela représente généralement moins d'une journée de prestation et cela risque de lui en faire gagner bien davantage.

Ceux qui ont vraiment goûté à cet EDI, comme moi, n'ont plus envie de faire machine arrière et ne

peuvent plus se passer de l'outil. Lors de la sortie de l'édition de 2013, j'avais justement écrit un article à ce sujet. Je vous invite donc à le parcourir ([lien 44](#)) pour comprendre ce qui donne à IntelliJ IDEA sa notoriété. J'y parle notamment de l'autocomplétion, de sa puissance d'analyse ainsi que de ses inspecteurs, des outils de *refactoring*, ou encore du « SSR » (ou « Structural Search and Replace »), fonctionnalité ultra puissante, mais difficile à manipuler.

4 Petits conseils pour bien migrer vers IntelliJ IDEA

Fin 2012, j'ai acheté la version 12 d'IntelliJ, bien décidé à m'y mettre pour de vrai. J'avais déjà essayé par le passé d'y jeter un œil, mais mes habitudes sur Eclipse ont eu raison de toutes mes tentatives. Mais cette fois-ci fut la bonne, et je ne voudrais jamais avoir à faire machine arrière ! Pour vous aider à franchir le pas, voilà quelques petites astuces.

Tout d'abord, il faut persévérer un peu. C'est vrai qu'il est parfois déstabilisant de quitter un outil - en l'occurrence Eclipse - dans lequel on a pris ses habitudes au fil des ans, mais croyez-moi, cette fois c'est pour votre bien !

Premier point : la vue « Workspace » d'Eclipse disparaît. IDEA ne gère qu'un seul projet à la fois par fenêtre (il est toutefois possible d'ouvrir autant de fenêtres d'IDEA que nécessaire). Finalement, ce n'est guère gênant, sauf si on a l'habitude de travailler sur dix projets en *même temps* (mais c'est peut-être là un signe qu'il y a un problème, non ?).

Autre point, les raccourcis clavier. On ne peut pas travailler efficacement sur un outil sans en connaître les raccourcis clavier, du moins les principaux. IDEA propose une option pour faire correspondre autant que possible les raccourcis clavier à ceux d'Eclipse ou de NetBeans. Pour cela, il faut aller dans *Settings > Keymap* puis choisir *Eclipse* ou *NetBeans* dans le sélecteur *Keymaps*. Toutefois, je vous conseille vivement de laisser les raccourcis

par défaut et de les apprendre. Pour vous aider, vous pouvez télécharger et imprimer un pense-bête ([lien 45](#)) (la version pour Mac : [lien 46](#)) ou pourquoi pas vous acheter un t-shirt ([lien 47](#)) ...

Il existe aussi le raccourci « universel », *Ctrl + Shift + A*, qui vous permet d'exécuter n'importe quelle action en tapant simplement son nom. Enfin, je vous recommande le plugin « *Key Promoter* » qui détecte quand vous réalisez une action alors qu'un raccourci clavier existe. Dans pareille situation, le plugin va afficher une popup vous indiquant le raccourci à utiliser pour gagner du temps. C'est très pratique.

Si certaines personnes de votre équipe travaillent toujours sur Eclipse (les pauvres !), alors il faudra peut-être penser à ajouter le plugin de formatage d'Eclipse ([lien 48](#)), ce qui vous assurera d'avoir les mêmes conventions que vos camarades. À noter aussi qu'il est possible de demander à IDEA de sauvegarder les métadonnées du projet au format Eclipse ([lien 49](#)) (fichier *.classpath*) plutôt qu'IDEA (fichier **.iml*).

Quoi qu'il en soit, si vous vous sentez perdu, n'hésitez pas à consulter la F.A.Q. ([lien 50](#)) ou le forum dédié de Developpez.com ([lien 51](#)).

D'autres conseils sur la page dédiée de JetBrains : [lien 52](#). Une page similaire existe si vous êtes un utilisateur de NetBeans : [lien 53](#).

5 Conclusion

À l'image de la version 13, cette nouvelle version n'est pas une révolution. JetBrains n'a cependant cessé d'améliorer son outil, le rendant toujours plus puissant.

Si le développement web n'a pas vos faveurs, alors il n'y a pas une seconde à perdre : jetez-vous sur la version *Community* qui est gratuite.

De même, si vous n'avez encore jamais mis les mains sur cet outil, alors il ne faut vraiment pas manquer l'occasion d'y jeter un œil, soit avec la ver-

sion **Community**, soit avec la version d'essai de 30 jours de la version **Ultimate**.

À noter qu'il existe un autre EDI développé par JetBrains : WebStorm ([lien 54](#)). Pour faire simple, il s'agit d'IntelliJ IDEA où toute la partie Java (disons plutôt tout ce qui a trait à la JVM) a été supprimée. Cet outil fait donc le bonheur des développeurs web, et a l'avantage d'être plus accessible (comptez 95 € pour l'édition personnelle).

6 Références

- Site officiel d'IDEA : [lien 55](#) ;
- Détails des fonctionnalités supportées par IDEA : [lien 56](#) ;
- Détails des nouveautés de la version 14 : [lien 57](#) ;
- FAQ d'IntelliJ IDEA sur Developpez.com : [lien 58](#) ;
- Article de présentation de la v13 : [lien 59](#) ;
- Forum IntelliJ de Developpez.com : [lien 60](#).

*Retrouvez l'article de **Romain Linsolas** en ligne : [lien 61](#)*

Eclipse



Les derniers tutoriels et articles

Tutoriel : À la découverte d'EMF.Edit

Avec EMF, l'Eclipse Modeling Framework, il est possible de générer du code rapidement pour un modèle donné. EMF ne se contente pas seulement de générer le code des classes, mais aussi d'instrumenter ce code. Grâce à cette instrumentation, il est possible de construire des frameworks de plus haut niveau, indépendants de la logique métier derrière un modèle. Parmi ces frameworks, EMF.Edit permet de générer et d'utiliser un ensemble de classes pour la visualisation et l'édition des modèles EMF. Cet article se propose de donner un aperçu des mécanismes de base de ce framework. Les sources de cet exemple sont disponibles à l'adresse suivante : FTP ([lien 62](#)) ou HTTP ([lien 63](#)).

1 Introduction

EMF, Eclipse Modeling Framework, permet de générer, un peu comme le ferait un générateur de code UML, une architecture de classes qui représentent un modèle métier. EMF ne se contente pas de générer seulement les classes Java, mais aussi toute une infrastructure associée. Ainsi, on bénéficiera par exemple de la persistance du modèle au format XMI, mais aussi d'un ensemble d'outils pour interroger le modèle de manière totalement indépendante des objets qu'il contient. Cette infrastructure permet de construire des outils de plus haut niveau pour traiter les modèles créés avec EMF. Au sein de ce framework, une des fonctionnalités est notamment la visualisation et l'édition de modèles grâce au framework EMF.Edit. Cet article se propose d'expliquer les mécanismes de base d'EMF.Edit et comment les adapter à des besoins propres. Nous verrons ainsi comment visualiser très rapidement l'intégralité d'un modèle et comment le modifier de manière

simple. Pour suivre cet article, il est indispensable de connaître comment créer et modifier un modèle EMF. Vous pouvez pour cela consulter les articles suivants :

- tutorial de Mickaël BARON : [lien 64](#) ;
- tutorial de Lars VOGEL : [lien 65](#).

Dans cet article, j'ai utilisé Eclipse Luna équipé des outils « Eclipse Modeling Framework SDK ». Nous allons suivre les étapes suivantes :

- Création du modèle EMF et options de génération ;
- Explication des mécanismes d'EMF.Edit pour la visualisation du modèle ;
- Explication des mécanismes d'EMF.Edit pour l'édition du modèle ;
- Databinding sur les propriétés des objets pour la construction d'IHM.

2 Création de l'infrastructure

2.1 Création du modèle EMF

La première étape est évidemment de créer un modèle EMF. Nous allons nous baser dans cet article sur un exemple simple, déjà utilisé dans l'article sur Sirius. Ce modèle représente les liaisons aériennes qui existent entre différents aéroports. Le modèle contiendra donc un certain nombre d'aéroports définis par un nom, une ville et un pays. Ces aéroports ont un ensemble de portes identifiées par un numéro. Ces portes peuvent référencer une porte d'un autre aéroport, représentant ainsi une liaison entre deux aéroports. Créez un nouveau projet de type « Ecore

Modeling Project » avec l'ID « com.abernard.airports ». Sur la deuxième page, le nom du package créé doit être « airports » et enfin sur la troisième page choisissez le viewpoint « Design » pour la création du diagramme. Cela va créer un fichier .ecore, un fichier .genmodel et ouvrir l'éditeur de diagrammes EMF. Créez ensuite chacun des éléments du modèle pour obtenir le diagramme suivant :

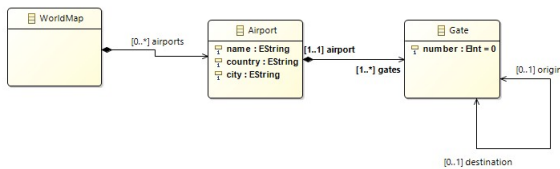


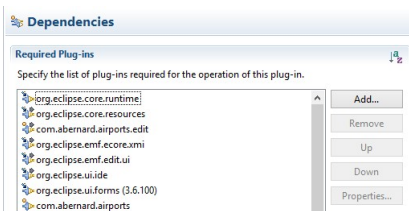
Diagramme du modèle EMF

2.2 Première génération

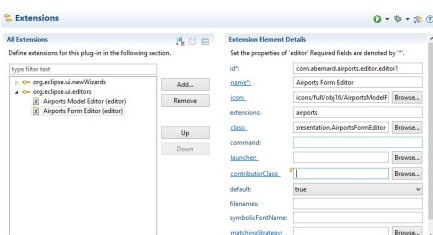
À partir du fichier *.genmodel, lancez la génération du plugin de modèle, ainsi que des plugins « Edit » et « Editor ». Eclipse va générer le code de toutes les classes ainsi que les plugins « com.abernard.airports.edit » et « com.abernard.airports.editor ». À partir de ce moment, vous pouvez lancer une instance d'Eclipse via le plugin « *.editor » et dès à présent créer un petit projet contenant un exemple d'instance de votre modèle

2.3 Création de l'infrastructure

Afin de tester les fonctionnalités d'EMF.Edit, nous allons créer une interface toute simple pour afficher notre modèle et l'éditer dans le plugin « com.abernard.airports.editor ». Afin de tirer parti des mécanismes Eclipse, nous allons utiliser un éditeur, plus spécifiquement un « FormEditor ». Ce type d'éditeur affiche des pages de type « Form », à la manière de l'éditeur de fichier « plugin.xml ». Il définit aussi un certain nombre d'implémentations par défaut afin d'éviter d'avoir à implémenter toutes les méthodes de l'interface « IEditorPart ». Pour cela, n'oubliez pas la dépendance au bundle « org.eclipse.ui.forms » dans le MANIFEST.MF :



Puis créez un éditeur qui hérite de « FormEditor » dans le point d'extension « org.eclipse.ui.editors ».



Dans cet éditeur, créez une unique page de type « FormPage » qui contiendra notre IHM. Vous pouvez utiliser les codes suivants afin de mettre en place ces interfaces vides.

```

1 package com.abernard.airports.
2   presentation;
3
4 import org.eclipse.core.runtime.
5   IProgressMonitor;
6 import org.eclipse.core.runtime.Status;
7 import org.eclipse.ui.IEditorInput;
8 import org.eclipse.ui.IEditorSite;
9 import org.eclipse.ui.PartInitException;
10 import org.eclipse.ui.forms.editor.
11   FormEditor;
12 import org.eclipse.ui.statushandlers.
13   StatusManager;
14
15 /**
16  * Cet éditeur contient une unique page
17  * pour éditer notre modèle EMF.
18  * @author A. BERNARD
19  */
20 public class AirportsFormEditor extends
21   FormEditor {
22
23   /* Étape 1 : mise en place */
24   private AirportsFormEditorPage
25     mainPage;
26
27   public AirportsFormEditor() {
28     //
29   }
30
31   @Override
32   protected void addPages() {
33     mainPage = new
34       AirportsFormEditorPage(this,
35         "com.abernard.airports.
36         presentation.form1", "
37         Contenu");
38     try {
39       addPage(mainPage);
40     } catch (PartInitException e) {
41       StatusManager.getManager().
42         handle(new Status(Status.
43           ERROR, "com.abernard.
44           airports.editor",
45             e.getMessage(), e),
46           StatusManager.SHOW);
47     }
48   }
49
50   @Override
51   public void doSave(IProgressMonitor
52     monitor) {
53   }
54
55   @Override
56   public void doSaveAs() {
57     // nous n'autorisons pas le
58     saveAs...
59   }
60
61   @Override
62   public void init(IEditorSite site,
63     IEditorInput input)
64     throws PartInitException {
65     super.init(site, input);
66   }
67
68   @Override
69   public boolean isSaveAsAllowed() {
70     return false;
71   }
72 }

```

```

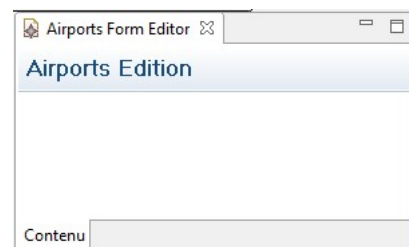
56 }
57 }

1 package com.abernard.airports.
  presentation;
2
3 import org.eclipse.swt.layout.FillLayout
  ;
4 import org.eclipse.ui.forms.IManagedForm
  ;
5 import org.eclipse.ui.forms.editor.
  FormPage;
6 import org.eclipse.ui.forms.widgets.
  FormToolkit;
7 import org.eclipse.ui.forms.widgets.
  ScrolledForm;
8
9 /**
10  * Cette unique page affiche notre modè
  le EMF dans un arbre et permet son
  édition.
11  * @author A. BERNARD
12  *
13  */
14 public class AirportsFormEditorPage
  extends FormPage {
15
16     private AirportsFormEditor editor;
17
18     public AirportsFormEditorPage(
  AirportsFormEditor
  airportsFormEditor,
19     String id, String title) {
20     super(airportsFormEditor, id,
  title);
  
```

```

21     this.editor = airportsFormEditor
  ;
22 }
23
24 @Override
25 protected void createFormContent(
  IManagedForm managedForm) {
26     FormToolkit toolkit =
  managedForm.getToolkit();
27     ScrolledForm scrolledForm =
  managedForm.getForm();
28     scrolledForm.setText("Airports
  Edition");
29     toolkit.decorateFormHeading(
  scrolledForm.getForm());
30     managedForm.getForm().getBody().
  setLayout(new FillLayout());
31 }
32 }
  
```

Si vous lancez votre application et ouvrez votre modèle avec ce nouvel éditeur, vous verrez une interface vide que nous allons utiliser par la suite.



Notre éditeur, vide

3 Visualisation de notre modèle

3.1 Visualisation « par défaut »

Nous allons maintenant visualiser notre modèle avec les outils de visualisation d'EMF.Edit par défaut. Regardons de plus près le code présent dans notre plugin « *.edit ». Une des classes se nomme « AirportsEditPlugin », elle ne présente pas d'intérêt intrinsèque : il s'agit de l'Activator du plugin. Un ensemble de classes nommées « *ItemProvider » sont aussi générées. Ces classes peuvent être utilisées pour afficher les éléments du modèle dans des composants JFace via un mécanisme de délégation : les adapters. Un adapter permet de « faire comme si » un objet de type A était un objet de type B. Pour regrouper toutes ces classes et fournir la bonne implémentation à la demande, une dernière classe est générée : « AirportsItemProviderAdapterFactory ». Elle génère les classes mentionnées précédemment au besoin. Toutes ces classes contiennent du code généré, qui doit être considéré comme tel : il est inutile de le modifier et nous verrons plus tard comment le faire de manière adaptée. Ce qu'il faut retenir, c'est que ces classes vont nous éviter beaucoup de codage manuel ! Voyons ça dans un exemple.

Nous voulons visualiser notre modèle dans un arbre, donc un TreeViewer. La première étape est de

charger notre modèle à partir de notre fichier .airports. Dans ce premier exemple, nous allons utiliser des mécanismes bas niveau d'EMF. Enrichissez la classe « AirportsFormEditor » comme suit :

```

1 package com.abernard.airports.
  presentation;
2
3 import java.util.Map;
4
5 import org.eclipse.core.resources.IFile;
6 import org.eclipse.core.runtime.
  IProgressMonitor;
7 import org.eclipse.core.runtime.Status;
8 import org.eclipse.emf.common.util.URI;
9 import org.eclipse.emf.ecore.resource.
  Resource;
10 import org.eclipse.emf.ecore.resource.
  ResourceSet;
11 import org.eclipse.emf.ecore.resource.
  impl.ResourceSetImpl;
12 import org.eclipse.emf.ecore.xmi.impl.
  XMIResourceFactoryImpl;
13 import org.eclipse.ui.IEditorInput;
14 import org.eclipse.ui.IEditorSite;
15 import org.eclipse.ui.IFileEditorInput;
16 import org.eclipse.ui.PartInitException;
17 import org.eclipse.ui.forms.editor.
  FormEditor;
18 import org.eclipse.ui.statushandlers.
  StatusManager;
19
  
```

```

20 import com.abernard.airports.WorldMap;
21
22 /**
23  * Cet éditeur contient une unique page
24  * pour éditer notre modèle EMF.
25  * @author A. BERNARD
26  */
27 public class AirportsFormEditor extends
28     FormEditor {
29
30     /* Étape 1 : mise en place */
31     private AirportsFormEditorPage
32         mainPage;
33     /* Étape 2 : chargement du modele */
34     private WorldMap myModel;
35
36     // ...
37
38     @Override
39     public void init(IEditorSite site,
40         IEditorInput input) throws
41         PartInitException {
42         super.init(site, input);
43         loadModel();
44     }
45
46     /**
47     * Charge le modèle à partir du
48     * fichier ouvert.
49     * @throws PartInitException
50     */
51     private void loadModel() throws
52         PartInitException {
53         /* Etape 2 : chargement du
54         modele */
55         if (getEditorInput() instanceof
56             IFileEditorInput) {
57             IFile inputFile = ((
58                 IFileEditorInput)
59                 getEditorInput()).
60                 getFile();
61             Resource.Factory.Registry
62                 reg = Resource.Factory.
63                 Registry.INSTANCE;
64             Map<String, Object> m = reg.
65                 getExtensionToFactoryMap
66                 ();
67             m.put("airports", new
68                 XMIResourceFactoryImpl()
69                 );
70
71             ResourceSet resSet = new
72                 ResourceSetImpl();
73             Resource resource = resSet.
74                 getResource(URI
75                 .createFileURI(inputFile
76                 .getLocation().
77                 toString()), true);
78             myModel = (WorldMap)
79                 resource.getContents().
80                 get(0);
81         } else {
82             throw new PartInitException(
83                 new Status(Status.ERROR,
84                     "com.abernard.airports.
85                     editor",
86                     "Unable to open resource
87                     "));
88         }
89     }
90
91     /**

```

```

65     * Retourne le modèle chargé dans
66     * cet éditeur.
67     * @return
68     */
69     WorldMap getModel() {
70         return myModel;
71     }

```

Dans notre page, il suffit ensuite de créer un `TreeViewer` qui va nécessiter un « `IContentProvider` » ainsi qu'un « `IBaseLabelProvider` ». Si nous travaillions sur des objets classiques, nous devrions écrire à la main toutes les méthodes pour parcourir de manière hiérarchique notre modèle ainsi que la manière d'afficher les éléments. `EMF.Edit` va ici entrer en jeu. La première étape est d'instancier la classe conteneur « `AirportsItemProviderAdapterFactory` », par exemple dans le constructeur de la classe. Puis nous pouvons utiliser cet objet pour instancier deux classes : « `AdapterFactoryContentProvider` » et « `AdapterFactoryLabelProvider` ». Ces classes utilisent notre « `adapter factory` » afin d'indiquer à notre composant comment afficher notre modèle. C'est tout. Modifiez le code de la classe « `AirportsFormEditorPage` » de la manière suivante et observez le résultat !

```

1 package com.abernard.airports.
2     presentation;
3
4 import org.eclipse.emf.edit.ui.provider.
5     AdapterFactoryContentProvider;
6 import org.eclipse.emf.edit.ui.provider.
7     AdapterFactoryLabelProvider;
8 import org.eclipse.jface.viewers.
9     TreeViewer;
10 import org.eclipse.swt.SWT;
11 import org.eclipse.swt.layout.FillLayout
12     ;
13 import org.eclipse.swt.layout.GridData;
14 import org.eclipse.swt.widgets.Composite
15     ;
16 import org.eclipse.swt.widgets.Tree;
17 import org.eclipse.ui.forms.IManagedForm
18     ;
19 import org.eclipse.ui.forms.editor.
20     FormPage;
21 import org.eclipse.ui.forms.widgets.
22     FormToolkit;
23 import org.eclipse.ui.forms.widgets.
24     ScrolledForm;
25
26 import com.abernard.airports.provider.
27     AirportsItemProviderAdapterFactory;
28
29 /**
30  * Cette unique page affiche notre modè
31  * le EMF dans un arbre et permet son
32  * édition.
33  * @author A. BERNARD
34  */
35 public class AirportsFormEditorPage
36     extends FormPage {
37
38     private AirportsFormEditor editor;
39     private TreeViewer viewer;
40
41     private

```

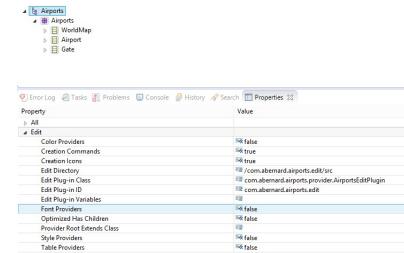

3.2 Les options de génération

```

29     adapterFactory;
30
31     public AirportsFormEditorPage(
32         AirportsFormEditor
33         airportsFormEditor,
34         String id, String title) {
35         super(airportsFormEditor, id,
36             title);
37         this.editor = airportsFormEditor
38             ;
39
40         this.adapterFactory = new
41             AirportsItemProviderAdapterFactory
42             ();
43
44     }
45
46     @Override
47     protected void createFormContent(
48         IManagedForm managedForm) {
49         FormToolkit toolkit =
50             managedForm.getToolkit();
51         ScrolledForm scrolledForm =
52             managedForm.getForm();
53         scrolledForm.setText("Airports
54             Edition");
55         toolkit.decorateFormHeading(
56             scrolledForm.getForm());
57         managedForm.getForm().getBody().
58             setLayout(new FillLayout());
59
60         Composite container =
61             managedForm.getForm().
62             getBody();
63         viewer = new TreeViewer(
64             container, SWT.BORDER | SWT.
65             MULTI);
66         Tree tree = viewer.getTree();
67         tree.setLayoutData(new GridData(
68             SWT.FILL, SWT.FILL, true,
69             true, 3, 1));
70         managedForm.getToolkit().
71             paintBordersFor(tree);
72         viewer.setContentProvider(new
73             AdapterFactoryContentProvider(
74                 adapterFactory));
75         viewer.setLabelProvider(new
76             AdapterFactoryLabelProvider(
77                 adapterFactory));
78         viewer.setInput(editor.getModel
79             ());
80     }
81 }

```

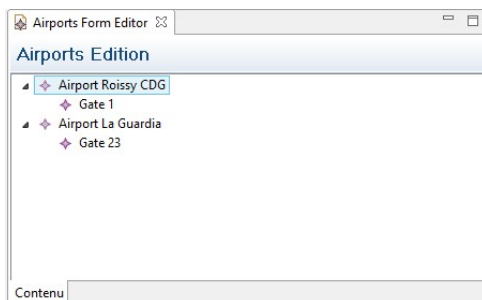
Tous les éléments que nous avons utilisés dans ce paragraphe ont été générés automatiquement dans le plugin *.edit de notre application. Vous pouvez évidemment modifier les options de génération pour personnaliser ces éléments. Ces options doivent être modifiées dans le fichier « airports.genmodel ». Les options plus générales sont définies dans l'élément racine du fichier, comme le montre la capture d'écran ci-dessous :



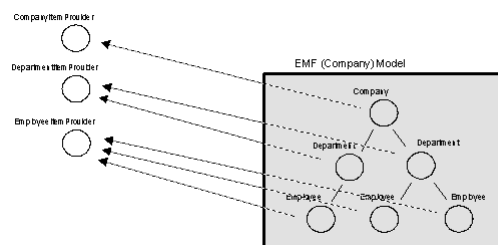
Options générales d'édition

Ces options vous permettent notamment de définir les différents « providers » JFace à générer. Par exemple, la propriété « Style Providers » vous permettra, après génération, d'accéder aux méthodes « getStyledText » afin d'afficher des styles sur vos labels. L'option « Table providers », elle, vous permettra de spécifier le nom des colonnes dans un tableau.

Si vous sélectionnez des éléments du modèle, vous pouvez définir des propriétés pour chacun, comme le fait d'associer une image aux éléments ou l'attribut à utiliser par défaut pour afficher les éléments. Un autre élément intéressant est le type de provider : cette propriété peut être définie comme « singleton » ou « stateful ». Avec un provider de type « singleton », un seul provider sera instancié et utilisé pour toutes les instances d'un objet de votre modèle :



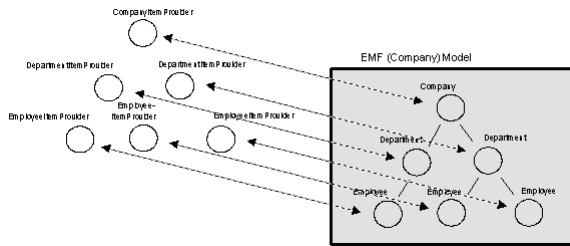
Visualisation de notre modèle



Singleton pattern

Remarquez qu'en instanciant simplement notre « adapter factory », nous avons pu l'utiliser pour afficher notre modèle très simplement dans l'arbre !


À l'inverse, avec un provider « stateful », un provider sera instancié pour chaque instance des objets de votre modèle :




Stateful pattern

3.3 Modifier la visualisation

Voir son modèle rapidement c'est bien, adapter l'affichage pour le mettre à son goût c'est mieux ! Dans ce paragraphe, nous allons modifier la manière dont les éléments de notre modèle sont affichés : le label pour les aéroports affichera directement la ville et le pays, et lorsque des portes sont reliées à d'autres portes, cette information sera elle aussi directement visible. Une première solution consiste à modifier directement le code des providers afin de modifier la méthode « getText() ». Pour peu qu'on supprime l'annotation « @generated », EMF ne remplacera pas le code écrit à la main. Pour autant, une bonne pratique consiste à traiter le code généré comme s'il s'agissait de bytecode. L'astuce consiste donc à utiliser le design pattern « Decorator ».

 Cette méthode est directement issue d'une présentation faite à différentes EclipseCon par Mikaël Barbero dont vous trouverez le lien en fin d'article.

Nous pouvons donc créer un dossier de source « src-dec » dans notre plugin « com.abernard.airports.edit ».

 N'oubliez pas de modifier le fichier « build.properties » afin d'ajouter le dossier « src-dec » aux dossiers sources.

Nous allons commencer par créer un décorateur générique pour tous nos éléments, que nous allons appeler « ItemProviderAdapterDecorator » :

```

1 package com.abernard.airports.provider.
  decorator;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.eclipse.emf.common.notify.
  Adapter;
7 import org.eclipse.emf.common.notify.
  AdapterFactory;
8 import org.eclipse.emf.common.notify.
  Notifier;
9 import org.eclipse.emf.edit.provider.
  IEditingDomainItemProvider;
10 import org.eclipse.emf.edit.provider.
  IItemLabelProvider;
  
```

```

11 import org.eclipse.emf.edit.provider.
  IItemPropertySource;
12 import org.eclipse.emf.edit.provider.
  IStructuredItemContentProvider;
13 import org.eclipse.emf.edit.provider.
  ITreeItemContentProvider;
14 import org.eclipse.emf.edit.provider.
  ItemProviderDecorator;
15
16 /**
17  * Cette classe est un décorateur pour
  nos items providers.
18  * @author A. BERNARD
19  *
20  */
21 public class
  ItemProviderAdapterDecorator extends
  ItemProviderDecorator implements
  Adapter.Internal,
22  IEditingDomainItemProvider,
  IStructuredItemContentProvider,
23  ITreeItemContentProvider,
  IItemLabelProvider,
  IItemPropertySource {
24
25     private List<Notifier> targets;
26
27     public ItemProviderAdapterDecorator (
  AdapterFactory adapterFactory) {
28         super(adapterFactory);
29     }
30
31     public Notifier getTarget() {
32         if (targets == null || targets.
  isEmpty()) {
33             return null;
34         } else {
35             return targets.get(targets.
  size() - 1);
36         }
37     }
38
39     public void setTarget(Notifier
  newTarget) {
40         if (targets == null) {
41             targets = new ArrayList<&t;
  Notifier>();
42         }
43         targets.add(newTarget);
44     }
45
46     public void unsetTarget(Notifier
  oldTarget) {
47         if (targets != null) {
48             targets.remove(oldTarget);
49         }
50     }
51 }
  
```

Puis nous créons tout d'abord un décorateur « vide » : il servira à rediriger l'appel de toutes les méthodes vers leur implémentation par défaut pour tous les éléments que nous n'allons pas modifier.

```

1 package com.abernard.airports.provider.
  decorator;
2
3 /**
4  * Cette classe est un décorateur "vide"
  qui délègue tous les appels aux
5  * méthodes par défaut.
6  * @author A. BERNARD
7  *
8  */
  
```

```

9 public class ForwardingItemProviderAdapterDecorator 13
10 extends ItemProviderAdapterDecorator {
11 public
12     ForwardingItemProviderAdapterDecorator
13     (WorldMapDecoratorAdapterFactory
14     airportsDecoratorAdapterFactory 15
15     ) {
16     super (
17     airportsDecoratorAdapterFactory18
18     );
19 }
20 }

```

Nous pouvons ensuite créer les deux décorateurs pour nos objets « Airport » et « Gate ». Ces deux classes se contentent de redéfinir l'implémentation de la méthode « getText() ».

```

1 package com.abernard.airports.provider.
2 decorator;
3 import com.abernard.airports.Airport;
4 /**
5  * Cette classe redéfinit la méthode {
6  * @link #getText(Object)} générée par
7  * défaut.
8  * @author A. BERNARD
9  */
10 public class AirportItemProviderDecorator extends
11     ItemProviderAdapterDecorator {
12     public AirportItemProviderDecorator(
13     WorldMapDecoratorAdapterFactory
14     airportsDecoratorAdapterFactory)
15     {
16     super(
17     airportsDecoratorAdapterFactory
18     );
19 }
20 @Override
21 public String getText(Object object)

```

```

1 package com.abernard.airports.provider.
2 decorator;
3 import com.abernard.airports.
4     AirportsPackage;
5 import com.abernard.airports.Gate;
6 /**
7  * Cette classe redéfinit la méthode {
8  * @link #getText(Object)} générée par
9  * défaut.
10 * @author A. BERNARD
11 */
12 public class GateItemProviderDecorator
13     extends ItemProviderAdapterDecorator
14     {

```

```

13     public GateItemProviderDecorator(
14     WorldMapDecoratorAdapterFactory
15     worldmapDecoratorAdapterFactory)
16     {
17     super(
18     worldmapDecoratorAdapterFactory
19     );
20 }
21 @Override
22 public String getText(Object object)
23 {
24     Gate gate = (Gate)object;
25     StringBuilder builder = new
26     StringBuilder("Gate ");
27     builder.append(gate.getNumber());
28     ;
29     if (gate.eIsSet(AirportsPackage.
30     eINSTANCE.
31     getGate_Destination())) {
32     builder.append(" -> ");
33     builder.append(gate.
34     getDestination().
35     getAirport().getName());
36     builder.append(" (");
37     builder.append(gate.
38     getDestination().
39     getNumber());
40     builder.append(")");
41 }
42     return builder.toString();
43 }

```

Enfin, il ne reste plus qu'à agréger tous ces éléments au sein d'une « adapter factory » qui réutilisera évidemment la classe par défaut « AirportsItemProviderAdapterFactory » :

```

1 package com.abernard.airports.provider.
2 decorator;
3 import org.eclipse.emf.edit.provider.
4     DecoratorAdapterFactory;
5 import org.eclipse.emf.edit.provider.
6     IItemProviderDecorator;
7 import com.abernard.airports.Airport;
8 import com.abernard.airports.Gate;
9 import com.abernard.airports.provider.
10     AirportsItemProviderAdapterFactory;
11 /**
12  * Cette nouvelle classe remplace celle
13  * par défaut {@link
14  * AirportsItemProviderAdapterFactory}
15  * afin de rediriger les appels vers nos
16  * nouveaux items providers.
17  * @author A. BERNARD
18  */
19 public class WorldMapDecoratorAdapterFactory
20     extends DecoratorAdapterFactory {
21     public
22     WorldMapDecoratorAdapterFactory
23     () {
24     super(new
25     AirportsItemProviderAdapterFactory
26     ());
27 }

```

```

22 @Override
23 protected IItemProviderDecorator
   createItemProviderDecorator(
   Object target, Object Type) {
24     if (target instanceof Airport) {
25         return new
   AirportItemProviderDecorator(
   (this));
26     } else if (target instanceof
   Gate) {
27         return new
   GateItemProviderDecorator
   (this);
28     }
29     return new
   ForwardingItemProviderDecorator
   (this);
30 }
31
32 }

```

Il ne nous reste plus qu'à observer le résultat. Dans la classe « AirportsFormEditorPage », il suffit de modifier l'instanciation de l'« adapter factory » avec notre nouvelle classe (et de modifier le type de l'attribut en conséquence).

```

1 // ...
2 this.adapterFactory = new
   WorldMapDecoratorAdapterFactory ();
3 // ...

```



Nouveaux labels

On l'a vu, le système des « adapter factories » permet d'afficher rapidement les éléments d'un modèle. Mais on peut aller encore plus loin, en combinant plusieurs factories en une seule pour afficher au sein d'un même viewer plusieurs modèles EMF différents. Pour cela on passe par une classe spécifique, la « ComposedAdapterFactory ». Une fois instanciée,

4 Édition du modèle

Un des énormes avantages d'EMF.Edit est que le framework introduit des outils pour faciliter la manipulation et l'édition des modèles. Ces outils sont principalement la « CommandStack » et « l'EditingDomain ». Le premier consiste en une « pile de commandes » littéralement qui sont stockées lors de leur exécution, afin d'être éventuellement annulées par la suite. Elles sont principalement constituées de trois catégories principales : les commandes de type « Add » (pour ajouter des éléments à un modèle), les commandes de type « Remove » (pour supprimer des

on peut lui ajouter toutes celles que l'on veut traiter. Par exemple, dans l'éditeur généré par défaut par EMF, on a le code suivant dans la méthode « initializeEditingDomain() » :

```

1 // ...
2 protected void initializeEditingDomain()
   {
3     // Create an adapter factory that
   yields item providers.
4     //
5     adapterFactory = new
   ComposedAdapterFactory(
   ComposedAdapterFactory.
   Descriptor.Registry.INSTANCE);
6
7     adapterFactory.addAdapterFactory(new
   ResourceItemProviderAdapterFactory
   ());
8     adapterFactory.addAdapterFactory(new
   AirportsItemProviderAdapterFactory
   ());
9     adapterFactory.addAdapterFactory(new
   ReflectiveItemProviderAdapterFactory
   ());
10 // ...
11 }

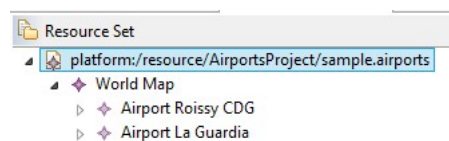
```

C'est ce mécanisme qui permet entre autres à l'éditeur par défaut de combiner l'affichage de la « Resource » contenant le modèle avec le modèle lui-même. Notez que si vous voulez accéder à une adapter factory capable d'afficher n'importe quel modèle disponible dans votre plateforme courante, vous pouvez utiliser l'instruction suivante :

```

1 ComposedAdapterFactory
   composedAdapterFactory = new
   ComposedAdapterFactory(
   ComposedAdapterFactory.Descriptor.
   Registry.INSTANCE);

```



Composed Factory

éléments du modèle) et enfin des commandes de type « Set » (pour modifier des attributs du modèle). Ces commandes peuvent être combinées au sein de commandes appelées « CompoundCommand ». L'objet « EditingDomain » quant à lui permet d'accéder au modèle afin de l'éditer. Il permet d'instancier les commandes citées précédemment, de les exécuter et enfin de faciliter le chargement et la sauvegarde du modèle. Nous allons voir comment mettre en place et utiliser ces éléments sur notre modèle. La première étape consiste à initialiser l'EditingDomain et

la CommandStack dans notre éditeur, puis à s'en servir pour charger notre modèle. Nous allons remplacer la méthode « loadModel » par deux autres, « initializeEditingDomain » et « loadModelWithDomain ». Toutes ces méthodes sont donc dans la classe « AirportsFormEditor.java ».

```

1 // imports ...
2 public class AirportsFormEditor extends
   FormEditor implements
   IEditingDomainProvider {
3
4   /* Étape 3 : tester un composed
   adapter */
5   private ComposedAdapterFactory
   adapterFactory;
6   /* Étape 4 : créer un EditingDomain
   */
7   private AdapterFactoryEditingDomain
   editingDomain;
8
9   // ...
10
11  @Override
12  public void doSave(IProgressMonitor
   monitor) {
13  editingDomain.doSave(monitor);
14  }
15
16  @Override
17  public void init(IEditorSite site,
   IEditorInput input)
18  throws PartInitException {
19  super.init(site, input);
20  // loadModel();
21  initializeEditingDomain();
22  loadModelWithDomain();
23  }
24
25  /**
26   * Sauvegarde le modèle en utilisant
   les informations contenues
   dans l' {@link EditingDomain}.
27   * @param monitor
28   */
29  private void editingDomainDoSave(
   IProgressMonitor monitor) {
30  final Map<Object, Object>
   saveOptions = new HashMap<
   Object, Object>();
31  saveOptions.put(Resource.
   OPTION_SAVE_ONLY_IF_CHANGED,
   Resource.
   OPTION_SAVE_ONLY_IF_CHANGED_MEMORY);
32  saveOptions.put(Resource.
   OPTION_LINE_DELIMITER,
   Resource.
   OPTION_LINE_DELIMITER_UNSPECIFIED);
33
34  // Utiliser une "
   WorkspaceModifyOperation"
   est tout indiqué pour une
   action qui peut
35  // être longue au sein du
   workspace
36  WorkspaceModifyOperation
   operation = new
   WorkspaceModifyOperation() {
37  @Override
38  public void execute(
   IProgressMonitor monitor
   ) {
39  Resource resource =
   editingDomain.
   getResourceSet().
   getResources().get(0);
40  if (!editingDomain.
   isReadOnly(resource)
   ) {
41  try {
42  resource.save(
   saveOptions);
43  } catch (Exception
   exception) {
44  AirportsEditorPlugin
   .INSTANCE.
   log(
   exception);
45  }
46  }
47  }
48  };
49
50  try {
51  new ProgressDialog(
   getSite().getShell()).
   run(true, false,
   operation);
52  ((BasicCommandStack)
   editingDomain.
   getCommandStack()).
   saveIsDone();
53  firePropertyChange(
   IEditorPart.PROP_DIRTY);
54  } catch (Exception exception) {
55  AirportsEditorPlugin.
   INSTANCE.log(
   exception);
56  }
57  }
58  /**
59   * Charge le modèle en utilisant l' {@link
   EditingDomain}
60   */
61  private void loadModelWithDomain() {
62  URI resourceURI = EditUIUtil.
   getURI(getEditorInput());
63  Resource resource =
   editingDomain.getResourceSet().
   getResource(resourceURI,
   true);
64  myModel = (WorldMap) resource.
   getContents().get(0);
65  }
66  /**
67   * Initialise la {@link CommandStack}
   et l' {@link EditingDomain} en
   utilisant
68   * une {@link ComposedAdapterFactory}
69   */
70  private void initializeEditingDomain() {
71  adapterFactory = new
   ComposedAdapterFactory();
72  adapterFactory.addAdapterFactory(
   new
   WorldMapDecoratorAdapterFactory());
73  }
74
75

```

```

76     adapterFactory.addAdapterFactory
       (new
       ResourceItemProviderAdapterFactory
       ());
77     adapterFactory.addAdapterFactory
       (new EcoreAdapterFactory());
78     adapterFactory.addAdapterFactory
       (new
       ReflectiveItemProviderAdapterFactory
       ());
79
80     BasicCommandStack commandStack =
       new BasicCommandStack();
81     commandStack.
       addCommandStackListener(new
       CommandStackListener() {
82         @Override
83         public void
           commandStackChanged(
           EventObject event) {
84             Display.getDefault().
               asyncExec(new
               Runnable() {
85                 @Override
86                 public void run() {
87                     firePropertyChange
                       (PROP_DIRTY)
                       ;
88                 }
89             });
90         }
91     });
92     editingDomain = new
       AdapterFactoryEditingDomain(
       adapterFactory, commandStack
       );
93 }
94
95 /**
96  * Retourne l'instance de {@link
       ComposedAdapterFactory} utilisé
       e dans cet éditeur.
97  * @return
98  */
99     ComposedAdapterFactory
       getAdapterFactory() {
100         return adapterFactory;
101     }
102
103     @Override
104     public EditingDomain
       getEditingDomain() {
105         return editingDomain;
106     }
107
108     @Override
109     public boolean isDirty() {
110         boolean partsDirty = super.
           isDirty();
111         boolean emfDirty = ((
           BasicCommandStack)
           editingDomain.
           getCommandStack()).
           isSaveNeeded();
112         return partsDirty || emfDirty;
113     }
114     // ...
115 }

```

Dans le code que nous avons modifié, plusieurs points importants sont à noter. Intéressons-nous tout d'abord à la méthode « initializeEditingDomain ». Nous utilisons une implémentation par défaut

de l'interface « CommandStack », « BasicCommandStack » qui suffira à la plupart des besoins. De la même manière, nous utilisons pour l'interface « EditingDomain » une implémentation standard « AdapterFactoryEditingDomain ». Cette implémentation utilise les adapter factories que nous avons mentionnés dans les paragraphes précédents pour accéder aux éléments du modèle via les « *ItemProvider ». Nous ajoutons aussi un « CommandStackListener » afin d'informer le workbench à chaque fois qu'une commande est exécutée. Cela permettra la sauvegarde du modèle à chaque fois qu'il est modifié. Lorsque la propriété « PROP_DIRTY » est modifiée, le workbench appelle la méthode « isDirty » sur l'éditeur courant. Il faut donc aussi que nous modifiions cette fonction. Dans un FormEditor, nous pouvons déjà utiliser l'implémentation par défaut qui vérifie si un des onglets de l'éditeur est « dirty ». Nous ajoutons à cela un test sur notre CommandStack qui peut nous indiquer si des commandes ont été exécutées sur notre modèle depuis son état initial. Enfin lorsque nous effectuons la sauvegarde de notre modèle, il faut aussi penser à informer la CommandStack que la sauvegarde a été effectuée. Cela permet de faire en quelque sorte un « reset » sur cet élément. Une fois que ces mécanismes sont mis en place, il ne nous reste plus qu'à modifier notre interface pour ajouter des commandes. Nous allons tester les trois commandes au sein de trois actions simples. Pour cela, nous devons enrichir notre classe « AirportsEditorFormPage » pour l'ajout des boutons idoines.

```

1 package com.abernard.airports.
2     presentation;
3
4 // imports...
5 /**
6  * Cette unique page affiche notre modèle
       le EMF dans un arbre et permet son
       édition.
7  * @author A. BERNARD
8  *
9  */
10 public class AirportsFormEditorPage
       extends FormPage {
11
12     private AirportsFormEditor editor;
13     private TreeViewer viewer;
14
15     public AirportsFormEditorPage(
       AirportsFormEditor
       airportsFormEditor, String id,
       String title) {
16         super(airportsFormEditor, id,
           title);
17         this.editor = airportsFormEditor
           ;
18
19     }
20
21     @Override
22     protected void createFormContent(
       IManagedForm managedForm) {

```

```

23 FormToolkit toolkit =
    managedForm.getToolkit();
24 ScrolledForm scrolledForm =
    managedForm.getForm();
25 scrolledForm.setText("Airports
    Edition");
26 toolkit.decorateFormHeading(
    scrolledForm.getForm());
27 managedForm.getForm().getBody().
    setLayout(new FillLayout());
28 Composite container =
    managedForm.getForm().
    getBody();
29 container.setLayout(new
    GridLayout(3, false));
30
31 Button buttonNewAirport =
    managedForm.getToolkit().
    createButton(container, "New
    Airport", SWT.NONE);
32 buttonNewAirport.setLayoutData(
    new GridData(SWT.LEFT, SWT.
    CENTER, true, false, 1, 1));
33 buttonNewAirport.
    addSelectionListener(new
    SelectionAdapter() {
34 @Override
35 public void widgetSelected(
    SelectionEvent e) {
36     handleCreateNewAirport()
    ;
37 }
38 });
39 Button buttonRemoveAirport =
    managedForm.getToolkit().
    createButton(container, "
    Remove selected airports",
    SWT.NONE);
40 buttonRemoveAirport.
    setLayoutData(new GridData(
    SWT.LEFT, SWT.CENTER, true,
    false, 1, 1));
41 buttonRemoveAirport.
    addSelectionListener(new
    SelectionAdapter() {
42 @Override
43 public void widgetSelected(
    SelectionEvent e) {
44     handleRemoveAirport();
45 }
46 });
47
48 Button changeNameAirport =
    managedForm.getToolkit().
    createButton(container, "
    Change selection name", SWT.
    NONE);
49 changeNameAirport.setLayoutData(
    new GridData(SWT.LEFT, SWT.
    CENTER, true, false, 1, 1));
50 changeNameAirport.
    addSelectionListener(new
    SelectionAdapter() {
51 @Override
52 public void
    widgetSelected(
    SelectionEvent e) {
53     handleChangeNameAirport
    ();
54 }
55 });
56
57 viewer = new TreeViewer(
    container, SWT.BORDER | SWT.
    MULTI);
58 Tree tree = viewer.getTree();
59 tree.setLayoutData(new GridData(
    SWT.FILL, SWT.FILL, true,
    true, 3, 1));
60 managedForm.getToolkit().
    paintBordersFor(tree);
61 viewer.setContentProvider(new
    AdapterFactoryContentProvider(
    editor.getAdapterFactory())
    );
62 viewer.setLabelProvider(new
    AdapterFactoryLabelProvider(
    editor.getAdapterFactory())
    );
63 viewer.setInput(editor.getModel
    ());
64 getSite().setSelectionProvider(
    viewer);
65 }
66 /**
67  * Ajoute un aéroport au modèle en
68  * utilisant une {@link AddCommand
    }.
69  */
70 private void handleCreateNewAirport
    () {
71     Command add = AddCommand.create(
    editor.getEditingDomain(),
    editor.getModel(),
    AirportsPackage.eINSTANCE.
    getWorldMap_Airports(),
    AirportsFactory.
    eINSTANCE.createAirport
    ());
72     editor.getEditingDomain().
    getCommandStack().execute(
    add);
73 }
74 /**
75  * Supprime tous les aéroports sé
76  * lectionnés du modèle en
77  * utilisant une {@link
    RemoveCommand}.
78  */
79 private void handleRemoveAirport() {
    IStructuredSelection ssel = (
    IStructuredSelection)viewer.
    getSelection();
80     if (!sssel.isEmpty()) {
81         List<Airport>
            itemsToRemove = new
            ArrayList<>();
82         for (Iterator<?> it =
            ssel.iterator(); it.
            hasNext();) {
83             Object o = it.next();
84             if (o instanceof Airport
            ) {
85                 itemsToRemove.add((
            Airport)o);
86             }
87         }
88     }
89     Command c = RemoveCommand.
    create(editor.
    getEditingDomain(),
    itemsToRemove);
90     editor.getEditingDomain().
    getCommandStack().
    execute(c);
91 }

```

```

92     }
93 }
94
95 /**
96  * Change le nom de tous les aé
97  * roports sélectionnés en {@code
98  * Toto} en utilisant
99  * une {@link SetCommand}.
100 */
101 private void handleChangeNameAirport
102 () {
103     IStructuredSelection ssel = (
104     IStructuredSelection)viewer.
105     getSelection();
106     if (!sssel.isEmpty()) {
107         for (Iterator<?> it =
108             ssel.iterator(); it.
109             hasNext();) {
110             Object o = it.next();
111             if (o instanceof Airport
112             ) {
113                 Command set =
114                     SetCommand.
115                     create(editor.
116                     getEditingDomain
117                     (), (Airport)o,
118                     AirportsPackage.
119                     eINSTANCE.
120                     getAirport_Name
121                     (), "Toto");
122                 editor.
123                     getEditingDomain
124                     ().
125                     getCommandStack
126                     ().execute(set);
127             }
128         }
129     }
130 }

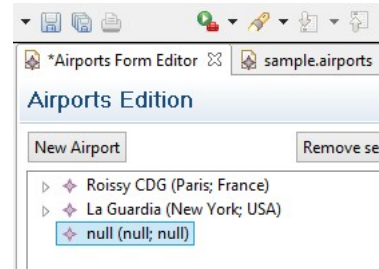
```

Nous avons ajouté trois boutons pour ajouter un aéroport, en supprimer et remplacer le nom par « Toto ». Tout d'abord la commande « Add ». Si on regarde la syntaxe de la commande, rien de transcendant. On communique l'EditingDomain afin que la commande puisse manipuler ledit modèle, le modèle en question, le nom de la « feature » à laquelle ajouter le nouvel élément ainsi que le nouvel élément créé. On peut s'en douter, cette commande va ajouter le nouvel élément créé à la fin de la liste « airports » de notre modèle. Puis, au lieu d'appeler simplement la méthode « execute » sur la commande, on passe par la CommandStack. Utiliser ces mécanismes a de multiples avantages :

- on peut annuler l'action en utilisant simplement « editingDomain.getCommandStack().undo() » ;
- on peut vérifier si un modèle doit être sauvegardé (rappelez-vous la méthode « isSaveNeeded » que nous avons utilisée dans la méthode « isDirty » de notre éditeur) ;
- on peut enfin utiliser l'instruction « myCommand.canExecute() » pour s'assurer que le contexte courant permet l'exécution d'une commande.


Notez que pour personnaliser le comportement des commandes, on peut écrire des sous-classes de toutes les commandes définies par le framework.

Les deux autres commandes fonctionnent exactement de la même manière et leur fonctionnement est aisément compréhensible. On peut vérifier le bon fonctionnement de ces commandes, et le fait que notre éditeur peut bien être sauvegardé lorsqu'on les a utilisées.



Test de la commande 'Add'

La cerise sur le gâteau est que l'utilisation de ces mécanismes (commandes et EditingDomain) va nous permettre de mettre en place des commandes de type « Undo/Redo » ou « Copy/Paste » simplement. Pour cela, il suffit de créer une barre d'outils d'éditeur qui est une sous-classe de « EditingDomainActionBarContributor » (pour rappel, cela se fait dans le champ « contributorClass » de votre éditeur dans le fichier « plugin.xml »). Cette classe active ses propriétés dès lors que l'éditeur courant implémente l'interface « IEditingDomainProvider ». On peut dès lors profiter de ces mécanismes sans autre action supplémentaire !



Pour que ces commandes fonctionnent correctement, il est impératif que l'arbre soit enregistré comme fournisseur de sélection, grâce à l'instruction « getSite().setSelectionProvider(viewer) ; ».

Comme d'habitude, on peut modifier le comportement de la classe, par exemple, on peut activer très simplement la validation de notre modèle en ajoutant dans le code la commande de validation. Les contraintes basiques des objets de notre modèle sont alors validées (comme dans l'éditeur Ecore).

```

1 package com.abernard.airports.
2 presentation;
3
4 import org.eclipse.emf.edit.ui.action.
5 EditingDomainActionBarContributor;
6 import org.eclipse.emf.edit.ui.action.
7 ValidateAction;
8 import org.eclipse.jface.action.
9 IToolBarManager;
10
11 /**
12  * L'héritage de {@link
13  * EditingDomainActionBarContributor}
14  * permet de bénéficier

```



```

9  * automatiquement des commandes Undo/
    Redo/etc.
10 * @author A. BERNARD
11 *
12 */
13 public class
    FormEditorActionBarContributor
    extends
    EditingDomainActionBarContributor {
14
15     public
        FormEditorActionBarContributor()
        {
16         //
17     }
18
19     @Override

```

```

20     public void contributeToToolBar(
        IToolBarManager toolBarManager)
        {
21         super.contributeToToolBar(
            toolBarManager);
22         validateAction = new ValidateAction
            ();
23         toolBarManager.add(validateAction);
24     }
25
26 }

```

Enfin, si on veut bénéficier de commandes plus avancées, comme le menu contextuel pour la création d'éléments mis en place dans l'éditeur généré, il suffit de réutiliser ou sous-classer la classe générée « AirportsActionBarContributor ».

5 Databinding avec EMF.Edit

5.1 Mise en place

Nous savons maintenant comment afficher les éléments contenus dans notre modèle et éventuellement les modifier. Néanmoins nous aurons vite le besoin ou l'envie de construire des interfaces plus spécifiques pour modifier les propriétés avancées. Nous allons voir dans ce paragraphe comment éditer les propriétés des objets EMF via le databinding. Ce mécanisme consiste à lier directement les valeurs affichées dans l'IHM aux valeurs contenues dans le modèle. Lorsque le modèle est modifié, l'affichage l'est aussi directement et vice-versa. Cela évite de passer par les classiques, mais fastidieux mécanismes de listeners, etc. Le databinding peut être réalisé sur des objets classiques, cf. ce tutoriel : [lien 66](#). Nous allons créer un élément de type MasterDetails pour afficher les détails des éléments sélectionnés dans l'arbre. Pour cela, il faut de nouveau compléter la vue « AirportsFormEditorPage ».

```

1  package com.abernard.airports.
    presentation;
2
3  // imports...
4
5  public class AirportsFormEditorPage
    extends FormPage {
6
7      // ...
8
9      @Override
10     protected void createFormContent(
        IManagedForm managedForm) {
11         FormToolkit toolkit = managedForm.
            getToolkit();
12         ScrolledForm scrolledForm =
            managedForm.getForm();
13         scrolledForm.setText("Airports
            Edition");
14         toolkit.decorateFormHeading(
            scrolledForm.getForm());
15         managedForm.getForm().getBody().
            setLayout(new FillLayout());
16         MasterDetailsBlock masterDetails =
            new MasterDetailsBlock() {

```

```

17
18         @Override
19         protected void registerPages(
            DetailsPart detailsPart) {
20             detailsPart.registerPage(
                AirportImpl.class, new
                AirportDetailsPage(editor));
21
22         }
23
24         @Override
25         protected void
            createToolBarActions(
                IManagedForm managedForm) {
26
27         }
28
29
30         @Override
31         protected void createMasterPart(
            final IManagedForm
            managedForm, Composite
            parent) {
32             final SectionPart sPart = new
                SectionPart(parent,
                managedForm.getToolkit(),
                Section.TITLE_BAR);
33             managedForm.addPart(sPart);
34             Composite container =
                managedForm.getToolkit().
                createComposite(sPart.
                getSection(), SWT.NONE);
35             sPart.getSection().setClient(
                container);
36             sPart.getSection().setText("
                WorldMap");
37
38             container.setLayout(new
                GridLayout(3, false));
39
40             Button buttonNewAirport =
                managedForm.getToolkit().
                createButton(container, "New
                Airport", SWT.NONE);
41
42             // ... création des boutons et
                du viewer...
43
44             viewer.
                addSelectionChangedListener(

```

```

    new
    ISelectionChangedListener()
    {
45     @Override
46     public void selectionChanged
        (SelectionChangedEvent
47         event) {
            managedForm.
                fireSelectionChanged(
                    sPart, event.
                        getSelection());
48     }
49     });
50     }
51     };
52     masterDetails.createContent(
        managedForm, managedForm.getForm
        ().getBody());
53
54     }
55
56     // ...
57 }

```

Ce code supplémentaire crée le composant MasterDetails, il ne nous reste plus qu'à créer une page de détails pour nos objets « Airport ».

```

1  package com.abernard.airports.
   presentation;
2
3  import org.eclipse.core.databinding.
   Binding;
4  import org.eclipse.core.databinding.
   UpdateValueStrategy;
5  import org.eclipse.core.databinding.
   observable.value.IObservableValue;
6  import org.eclipse.core.databinding.
   observable.value.WritableValue;
7  import org.eclipse.core.runtime.IStatus;
8  import org.eclipse.core.runtime.Status;
9  import org.eclipse.emf.databinding.
   EMFDataBindingContext;
10 import org.eclipse.emf.databinding.
   IEMFValueProperty;
11 import org.eclipse.emf.databinding.edit.
   EMFEditProperties;
12 import org.eclipse.jface.databinding.
   fieldassist.ControlDecorationSupport
   ;
13 import org.eclipse.jface.databinding.swt.
   IWidgetValueProperty;
14 import org.eclipse.jface.databinding.swt.
   WidgetProperties;
15 import org.eclipse.jface.viewers.
   ISelection;
16 import org.eclipse.jface.viewers.
   IStructuredSelection;
17 import org.eclipse.swt.SWT;
18 import org.eclipse.swt.layout.FillLayout
   ;
19 import org.eclipse.swt.layout.GridData;
20 import org.eclipse.swt.layout.GridLayout
   ;
21 import org.eclipse.swt.widgets.Composite
   ;
22 import org.eclipse.swt.widgets.Display;
23 import org.eclipse.swt.widgets.Label;
24 import org.eclipse.swt.widgets.Text;
25 import org.eclipse.ui.forms.IDetailsPage
   ;
26 import org.eclipse.ui.forms.IFormPart;
27 import org.eclipse.ui.forms.IManagedForm
   ;

```

```

28 import org.eclipse.ui.forms.widgets.
   FormToolkit;
29 import org.eclipse.ui.forms.widgets.
   Section;
30
31 import com.abernard.airports.Airport;
32 import com.abernard.airports.
   AirportsPackage;
33
34 /**
35  * Cette classe affiche certaines
   informations principales de nos
   objets {@link Airport} :
36  * le nom, le pays et la ville.
37  * @author A. BERNARD
38  *
39  */
40 public class AirportDetailsPage
   implements IDetailsPage {
41
42     private AirportsFormEditor refEditor
   ;
43     private IManagedForm managedForm;
44     private Text textName;
45     private Text textCity;
46     private Text textCountry;
47
48     private IObservableValue modelValue
   =new WritableValue();
49
50     public AirportDetailsPage(
   AirportsFormEditor editor) {
51         this.refEditor = editor;
52     }
53
54     @Override
55     public void initialize(IManagedForm
   form) {
56         this.managedForm = form;
57     }
58
59     @Override
60     public void dispose() {
61
62     }
63
64     @Override
65     public boolean isDirty() {
66         return false;
67     }
68
69     @Override
70     public void commit(boolean onSave) {
71
72     }
73
74     @Override
75     public boolean setFormInput(Object
   input) {
76         return false;
77     }
78
79     @Override
80     public void setFocus() {
81         //
82     }
83
84     @Override
85
86
87
88
89     @Override

```

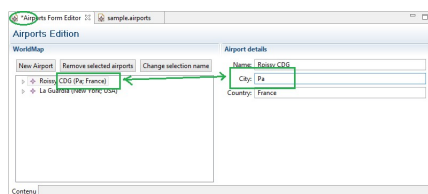
```

90     public boolean isStale() {
91         return false;
92     }
93
94     @Override
95     public void refresh() {
96         //
97     }
98
99     @Override
100    public void selectionChanged(
101        IFormPart part, ISelection
102        selection) {
103        IStructuredSelection ssel = (
104            IStructuredSelection)selection;
105        Object first = ssel.getFirstElement
106            ();
107        if (first != null &&& first
108            instanceof Airport) {
109            modelValue.setValue((Airport)
110                first);
111        }
112    }
113
114    @Override
115    public void createContents(Composite
116        parent) {
117        parent.setLayout(new FillLayout(SWT.
118            HORIZONTAL));
119        FormToolkit toolkit = managedForm.
120            getToolkit();
121        Section section = toolkit.
122            createSection(parent, Section.
123                TITLE_BAR);
124        section.setText("Airport details");
125        Composite container = toolkit.
126            createComposite(section);
127        container.setLayout(new GridLayout(2
128            , false));
129        section.setClient(container);
130
131        Label lblName = new Label(container,
132            SWT.NONE);
133        lblName.setLayoutData(new GridData(
134            SWT.RIGHT, SWT.CENTER, false,
135            false, 1, 1));
136        toolkit.adapt(lblName, true, true);
137        lblName.setText("Name:");
138
139        textName = new Text(container, SWT.
140            BORDER);
141        textName.setLayoutData(new GridData(
142            SWT.FILL, SWT.CENTER, true,
143            false, 1, 1));
144        toolkit.adapt(textName, true, true);
145
146        Label lblCity = new Label(container,
147            SWT.NONE);
148        lblCity.setLayoutData(new GridData(
149            SWT.RIGHT, SWT.CENTER, false,
150            false, 1, 1));
151        toolkit.adapt(lblCity, true, true);
152        lblCity.setText("City:");
153
154        textCity = new Text(container, SWT.
155            BORDER);
156        textCity.setLayoutData(new GridData(
157            SWT.FILL, SWT.CENTER, true,
158            false, 1, 1));
159        toolkit.adapt(textCity, true, true);
160
161        Label lblCountry = new Label(
162            container, SWT.NONE);
163
164        lblCountry.setLayoutData(new
165            GridData(SWT.RIGHT, SWT.CENTER,
166                false, false, 1, 1));
167        toolkit.adapt(lblCountry, true, true
168            );
169        lblCountry.setText("Country:");
170
171        textCountry = new Text(container,
172            SWT.BORDER);
173        textCountry.setLayoutData(new
174            GridData(SWT.FILL, SWT.CENTER,
175                true, false, 1, 1));
176        toolkit.adapt(textCountry, true,
177            true);
178
179        createDatabinding();
180    }
181
182    /**
183     * Initialise le databinding entre l
184     * 'élément du modèle et les
185     * composants graphiques.
186     */
187    private void createDatabinding() {
188        EMFDataBindingContext ctx = new
189            EMFDataBindingContext();
190        IWidgetValueProperty widgetValue =
191            WidgetProperties.text(SWT.Modify
192            );
193
194        // Airport name
195        IEMFValueProperty shortProp =
196            EMFEditProperties.value(
197                refEditor.getEditingDomain(),
198                AirportsPackage.Literals.
199                    AIRPORT__NAME);
200        Binding b = ctx.bindValue(
201            widgetValue.observeDelayed(200,
202                textName),
203            shortProp.observeDetail(
204                modelValue));
205
206        // Airport city
207        shortProp = EMFEditProperties.value(
208            refEditor.getEditingDomain(),
209            AirportsPackage.Literals.
210                AIRPORT__CITY);
211        b = ctx.bindValue(widgetValue.
212            observeDelayed(200, textCity),
213            shortProp.observeDetail(
214                modelValue));
215
216        // Airport country
217        shortProp = EMFEditProperties.value(
218            refEditor.getEditingDomain(),
219            AirportsPackage.Literals.
220                AIRPORT__COUNTRY);
221        b = ctx.bindValue(widgetValue.
222            observeDelayed(200, textCountry)
223            ,
224            shortProp.observeDetail(
225                modelValue));
226    }

```

Dans cette classe, intéressons-nous aux méthodes « selectionChanged » et « createDatabinding » (le reste ne constitue que de l'interface pure et ne présente pas d'intérêt). La première encapsule l'élément sélectionné dans l'arbre dans un objet de type « Wri-

tableValue » qui va permettre de modifier et lire les propriétés à observer et va déclencher les événements lorsque ces valeurs sont modifiées (soit par le modèle, soit par l'interface). L'objet IObservableValue va notamment éviter des NullPointerException dans tous les sens si jamais l'objet du modèle est « null » et que les champs de l'interface ne peuvent donc rien afficher. La deuxième méthode va lier chaque champ texte à une propriété EMF. Cela se fait en utilisant les méthodes statiques de la classe EMFEditProperties, qui évidemment va utiliser un objet EditingDomain. Cela va permettre directement de bénéficier de la sauvegarde et de l'« Undo/Redo » ! Notez aussi l'emploi de la méthode « observeDelayed » avec un temps en millisecondes : au lieu de modifier le modèle à chaque fois que le champ texte est modifié, un timer sera utilisé (ici 200 ms). Cela évite un trop grand nombre d'événements lorsque l'utilisateur est en train de taper son texte. Avec ces mécanismes, on peut vite vérifier que notre modèle est bien mis à jour lorsqu'on modifie une des trois valeurs.



Databinding simple

5.2 Validation

Évidemment, l'intérêt d'une interface est aussi d'empêcher l'utilisateur de rentrer des valeurs abracadabrantes, pour cela on peut évidemment réutiliser les mécanismes de validation du databinding dans Eclipse sur nos éléments. Par exemple nous allons stipuler que chacune des trois données doit commencer par une majuscule. Nous utilisons donc un objet spécifique de type « UpdateValueStrategy » pour valider la donnée entrée par l'utilisateur avant qu'elle ne soit communiquée au modèle.

```

1 package com.abernard.airports.
  presentation;
2
3 // imports...
4
5 public class AirportDetailsPage
  implements IDetailsPage {
6
7     // ...
8
9     /**
10      * Initialise le databinding entre l
      'élément du modèle et les
      composants graphiques.
11
12      */
13     private void createDatabinding() {
14         EMFDataBindingContext ctx = new
            EMFDataBindingContext();
            IWidgetValueProperty widgetValue =
            WidgetProperties.text(SWT.Modify
            );
    
```

```

15
16 // Airport name
17 IEMFValueProperty shortProp =
18     EMFEditProperties.value(
19         refEditor.getEditingDomain(),
20         AirportsPackage.Literals.
21             AIRPORT__NAME);
22 Binding b = ctx.bindValue(
23     widgetValue.observeDelayed(200,
24     textName),
25     shortProp.observeDetail(
26     modelValue), new
27     CapitalizedFirstLetter(
28     textName), null);
29 ControlDecorationSupport.create(b,
30     SWT.TOP | SWT.LEFT);
31
32 // Airport city
33 shortProp = EMFEditProperties.value(
34     refEditor.getEditingDomain(),
35     AirportsPackage.Literals.
36     AIRPORT__CITY);
37 b = ctx.bindValue(widgetValue.
38     observeDelayed(200, textCity),
39     shortProp.observeDetail(
40     modelValue), new
41     CapitalizedFirstLetter(
42     textCity), null);
43 ControlDecorationSupport.create(b,
44     SWT.TOP | SWT.LEFT);
45
46 // Airport country
47 shortProp = EMFEditProperties.value(
48     refEditor.getEditingDomain(),
49     AirportsPackage.Literals.
50     AIRPORT__COUNTRY);
51 b = ctx.bindValue(widgetValue.
52     observeDelayed(200, textCountry)
53     ,
54     shortProp.observeDetail(
55     modelValue), new
56     CapitalizedFirstLetter(
57     textCountry), null);
58 ControlDecorationSupport.create(b,
59     SWT.TOP | SWT.LEFT);
60
61 }
62
63 /**
64  * Cette classe valide que les donn
65  * es entrées commencent par une
66  * majuscule.
67  * @author A. BERNARD
68  *
69  */
70 private class CapitalizedFirstLetter
71     extends UpdateValueStrategy {
72
73     private Text text;
74
75     public CapitalizedFirstLetter(Text
76     textField) {
77         this.text = textField;
78     }
79
80     @Override
81     public IStatus validateBeforeSet(
82     Object value) {
83         IStatus status;
84         if (value instanceof String) {
85             char first = ((String) value).
86                 charAt(0);
    
```

```

57     if (Character.isUpperCase(first)
58         ) {
59         status = Status.OK_STATUS;
60     } else {
61         status = new Status(Status.ERROR, "com.abernard.
        airports.presentation",
        "Text shall start with
        an uppercase letter
        ");
62     }
63     } else {
64     status = super.validateBeforeSet
        (value);
65     }
66     if (status.getSeverity() !=
        Status.OK) {
67     text.setBackground(Display.
        getDefault().getSystemColor(
        SWT.COLOR_RED));
68     } else {
69     text.setBackground(Display.
        getDefault().getSystemColor(
        SWT.COLOR_WHITE));
70     }
71     return status;
72 }

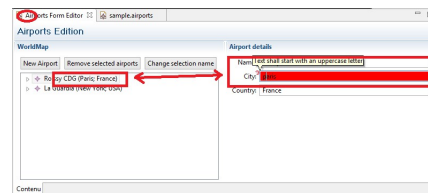
```

```

73     }
74 }

```

Notez l'utilisation très directe des objets « ControlDecorationSupport » qui vont afficher sur les champs le texte d'erreur ainsi qu'un petit marqueur visuel. On peut donc tester notre composant et vérifier que le modèle n'est pas modifié si nos valeurs ne commencent pas par une majuscule. Sur la capture ci-dessous, on voit bien le champ invalide avec la notification, on constate aussi que l'arbre n'est pas modifié (on a rentré « paris », mais le modèle contient toujours la valeur « Paris ») et l'éditeur n'est pas marqué comme « dirty ».



Databinding avec validation

6 Conclusion et perspectives

Dans cet article nous avons eu une première approche du framework EMF.Edit et j'espère qu'avec ces informations, vous aurez les éléments en main pour construire des interfaces autour de vos modèles EMF. Les mécanismes de ce framework permettent de mettre en œuvre rapidement une IHM avec tous les éléments nécessaires pour l'utilisateur final avec un minimum de code : validation des données, commandes classiques d'« Undo/Redo » ou de gestion du presse-papiers. En utilisant ce framework, on peut aussi aller encore plus loin et directement créer des

interfaces personnalisées sans écrire de code, grâce à des outils comme EMFForms, EMF Editing Framework ou encore EMFParsley. Ces outils feront peut-être l'objet de prochains articles ! En attendant, si vous pensez qu'un diagramme représenterait mieux notre modèle, vous pouvez jeter un œil à mon article sur Sirius : [lien 67](#). Enfin, l'éditeur généré par défaut ainsi que la barre d'outils associée, même s'ils peuvent être verbeux, sont un bon réservoir d'idées et de bonnes pratiques pour vos propres éditeurs !

7 Liens utiles

Vous trouverez dans cette section quelques liens qui peuvent être utiles sur EMF.Edit ou les frameworks évoqués en conclusion.

- EMF.Edit overview dans l'aide d'Eclipse : [lien 68](#) ;
- Présentation d'EMF.Edit : [lien 69](#) ;

- Tutoriel sur EMFForms : [lien 70](#) ;
- Extended Editing Framework : [lien 71](#) ;
- EMF Parsley : [lien 72](#).

N'oubliez pas que pour ce qui a trait à EMF, le livre « EMF, 2d Edition » reste une référence, bien qu'un peu daté !

Retrouvez l'article d'**Alain Bernard** en ligne : [lien 73](#)

Android



Les dernières news Android

Android 5.1 est disponible, avec quelques améliorations de moyenne importance

Après les correctifs de bogues détectés dans la version 5.0 de Lollipop suivis par la sortie des mises à jour 5.01 et 5.02, Google vient de publier à nouveau une mise à jour d'Android Lollipop avec comme numéro de référence « 5.1 ». L'annonce a été faite par le vice-président de la plateforme Android, Dave Burke. Elle a été éditée afin d'améliorer la stabilité et les performances du système et offrir par la même occasion de nouvelles fonctionnalités.

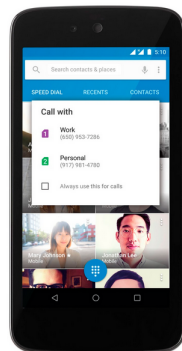
En guise de nouveautés apportées dans cette nouvelle référence, on note d'emblée la prise en charge native de plusieurs cartes SIM. Cela constitue un gain substantiel de temps pour les constructeurs qui implémentent des terminaux supportant deux ou trois puces. Au lieu de partir de zéro et concevoir tout le code qui permettra de gérer cet aspect, ils pourront s'appuyer sur cette base et apporter des modifications qu'ils souhaitent intégrer ou améliorer.

En second point, nous constatons une amélioration de la protection. Ainsi, si vous égarez votre terminal, le téléphone ou la tablette reste verrouillé tant que vous ne vous connectez pas à votre compte Google. Cette fonctionnalité reste active même après avoir effectué une réinitialisation de l'appareil aux paramètres d'usine. Vous pouvez donc souffler en cas de perte ou de vol, car les données de votre terminal seront protégées tant que l'accès au compte Google n'est pas corrompu. Cela semble assez intéressant et, espérons-le, permettra d'en dissuader plus d'un, même si on n'est pas à l'abri de l'installation d'une autre ROM sur l'appareil par un tiers – en cas de vol — afin de pouvoir l'utiliser. Cette caractéristique sera automatiquement disponible pour tous les équipements fonctionnant avec Android 5.1, de même que le Nexus 6 et 9.

Un autre ajout un peu exclusif cette fois-ci, c'est l'intégration de l'option d'appel en haute définition. Pour utiliser cette fonction, il faut avoir, en plus d'un appareil tournant avec Android 5.1, un opérateur de téléphonie supportant des communications avec une telle qualité sonore.

Enfin, nous notons en dernier point, un accès direct à l'option WiFi et Bluetooth depuis la fenêtre de configuration rapide.

Comme on peut le constater, cette mise à jour n'est pas une révolution en soi, mais a le mérite d'être présente afin d'améliorer l'expérience utilisateur.



Commentez la news d'**Olivier Famien** en ligne : [lien 74](#)

Google ne publiera plus de patchs de sécurité pour WebView dans les versions antérieures à Android 4.4 Kitkat, la sécurité de près d'un milliard de smartphones Android menacée

C'est sûrement une mauvaise nouvelle pour les utilisateurs d'Android qui exécutent des versions antérieures à Android 4.4 Kitkat et une bonne nouvelle pour les pirates qui auront plus de facilité.

Google a récemment pris la décision d'arrêter le développement de correctifs pour les bogues de WebView dans Android 4.3 Jelly Bean et les versions antérieures. Environ 60

WebView permet aux applications d'afficher des pages Web sans avoir à ouvrir une autre application. De nombreuses applications et des réseaux publicitaires utilisent le composant. D'ailleurs, l'équipe Google Android préconise l'outil dans sa documentation pour les développeurs sur le rendu des pages Web.

Toutefois, WebView est aussi le vecteur privilégié pour mener des attaques par exécution de code distant dans le système d'exploitation mobile, selon le directeur de l'ingénierie de Rapid7, Tod Beardsley. Des failles de logiciels ont, à plusieurs reprises, été découvertes dans Android et WebView, rendant le manque de mises à jour encore plus dangereux.

Après avoir reçu un rapport d'une nouvelle vulnérabilité dans le WebView antérieur à 4.4 en octobre 2014, les gestionnaires d'incidents Android de Google ont déclaré que la société va laisser aux développeurs externes la tâche de développer des correctifs de sécurité, d'après Beardsley. « *Si la version affectée [de WebView] est avant 4.4, nous ne développons généralement pas les patchs nous-mêmes, mais faisons bon accueil à des correctifs* » qui prennent en compte les défauts reportés. « *Nous ne sommes pas en mesure de mener des actions pour les défauts qui affectent les versions antérieures à 4.4 qui ne sont pas accompagnées d'un patch.* »

Cela suppose que la prochaine fois qu'un chercheur ou un pirate trouvera une vulnérabilité dans WebView sur une version Android antérieure à Kitkat, Google ne va pas, lui-même, créer un patch pour la vulnérabilité. Cependant, si quelqu'un d'autre en

développe, Google va intégrer ces correctifs dans le code Android Open Source Project. Google les donnera également aux fabricants de téléphones, mais c'est là que sa responsabilité s'arrête.



Android est open source, ce qui signifie techniquement que n'importe qui pourrait créer des patchs, mais les chances que ces correctifs soient distribués par les fabricants d'appareils comme Samsung sont minces, a ajouté Beardsley. La fin du support pour la plupart des utilisateurs d'Android pourrait donc augmenter le nombre d'attaques par téléchargements cachés sur Android.

Pour le directeur de l'ingénierie de Rapid7, la décision de supprimer le support de WebView pour les versions antérieures à Android Kitkat est suicidaire, mais elle trouve une explication dans les objectifs de Google. « *Bien sûr, j'espérais qu'abandonner le support de 60% de votre base d'installation serait suicidaire, cependant nous y sommes* », a-t-il dit.

Beardsley a ajouté que l'une des raisons pour lesquelles Google a décidé de ne plus fournir de correctifs de WebView pour Jelly Bean et les versions antérieures est que « *la meilleure façon de s'assurer que les appareils Android sont en sécurité est de les mettre à jour à la dernière version d'Android* », donc de passer par exemple à Android 5.0 Lollipop qui peine à convaincre : [lien 75](#). Il soupçonne, par ailleurs, que cette décision ait coïncidé avec la sortie de la dernière version d'Android.

Commentez la news de **Thibaut Cuvelier** en ligne : [lien 76](#)

Les derniers tutoriels et articles

Appel de procédure distante sous Android

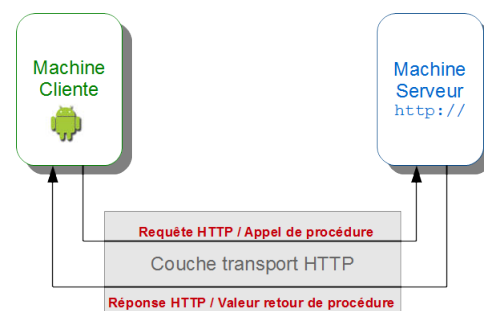
Une application mobile moderne fonctionne rarement en vase clos ; elle nécessite au contraire de dialoguer avec des applications tierces. Nous sommes donc en présence d'une application répartie dans laquelle l'application mobile constitue une partie cliente qui peut récupérer des données stockées sur, ou envoyer des données à, une partie serveur. Cet échange du client vers le serveur est recouvert par l'expression générique d'« appel de procédure distante », ou RPC. Ce tutoriel montre comment s'implémente ce mécanisme incontournable sous Android, à travers quelques exemples simples.

Les prérequis se trouvent dans le support de cours : lien 77.

1 Appel de procédure distante

C'est un terme générique pour décrire la situation où un programme fait appel à des procédures d'un autre programme situé dans un autre contexte. C'est typiquement le cas entre un programme tournant sur un périphérique Android nomade et un programme tournant sur un serveur localisé à des milliers de kilomètres de là. Il n'existe pas à proprement parler de technologie native d'appel de procédure distante sous Android (pas de RPC ou, plus spécifiquement, de RMI), mais il est possible d'obtenir une solution équivalente sans trop d'efforts. À mi-chemin entre les technologies SOAP ou XML-RPC (haut niveau, mais complexe à mettre en place) et les sockets (trop bas niveau), la solution la plus raisonnable est de se baser sur la couche transport réseau HTTP pour échanger des messages dont le format est totalement libre. Pour résumer, HTTP nous fournit un protocole fiable de type requête/réponse, le client et

le serveur n'auront qu'à se mettre d'accord sur les données échangées. L'avantage reste que cette solution est totalement indépendante des plates-formes et des langages côté serveur : on parle parfois de « web service » pour y faire référence.



Architecture d'appel de procédure distante via HTTP

2 Programmation client/serveur

S'adosser sur le protocole HTTP implique d'écrire des programmes dans des conteneurs web, c.-à-d. hébergés sur des serveurs HTTP (Apache HTTP, IIS, Node.js...), dans le langage de votre choix. L'autre aspect étant que tout est considéré comme une ressource, et par conséquent chaque procédure appellable côté serveur est incarnée par une URL (c'est à ce stade que l'on peut décider d'appliquer les principes d'une architecture web REST ou non).

Le présent tutoriel n'a pas pour objectif d'aborder les technologies web, une section dédiée à ces sujets étant par ailleurs disponible sur developpez.com. Retenez simplement que le but de tout programme web côté serveur est de produire une réponse HTTP quelconque à partir d'une requête HTTP du client, le plus souvent en exploitant le contenu d'une base

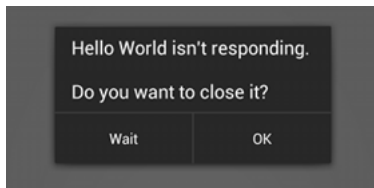
de données.

Dorénavant, concentrons-nous sur ce qui doit être fait au niveau du client Android.

2.1 Appel asynchrone

Historiquement, la technologie d'appel distant RPC repose sur des appels synchrones : le client reste bloqué dans l'attente de la réponse de serveur. Mais cela s'avère fortement handicapant dans notre contexte où le réseau possède un temps de latence, quand il n'est pas carrément défaillant. En effet, sur Android il est primordial que les appels soient asynchrones puisque le rendu graphique des écrans (Activity) de l'application est réalisé par un processus dédié, le *UI Thread*, qui ne doit pas être mis en attente d'une hypothétique réponse d'un serveur. Si

tel était le cas, cela signifierait un gel complet de l'interface utilisateur, ce qui n'est pas acceptable et déclenche typiquement une ANR (*Application Not Responding*).



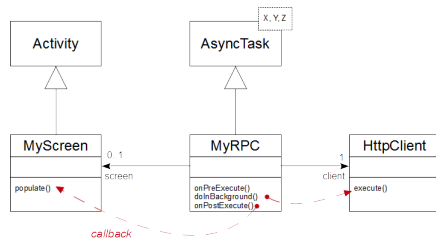
Le popup ANR affiché à l'utilisateur



Ce problème est un tel fléau, que depuis Android 3.0, une exception spéciale est levée : la *NetworkOnMainThreadException*.

2.2 Le patron de conception

La plupart du temps nous avons un écran graphique d'une part, et un appel de procédure distante asynchrone d'autre part, le tout devant être resynchronisé à un moment ou un autre. Il se trouve que l'API Android introduit un élément qui va grandement nous faciliter la vie, pour peu qu'on l'utilise correctement : la tâche asynchrone (*AsyncTask*). Je vous propose donc le patron de conception ci-dessous, à adapter selon vos besoins.



Patron de conception pour un RPC Android (Notation UML)

Selon ce patron, votre classe *MyScreen* implémente une méthode *populate()* dont le rôle exclusif est de mettre à jour les vues avec les données obtenues en retour de l'appel distant. L'instanciation du RPC est réalisée ici dans *onStart()* de sorte que chaque fois que l'écran prend le focus, l'appel distant est déclenché : idéal pour être certain d'avoir des données fraîches. Bien sûr, vous pouvez déplacer cette instruction ailleurs.

```

1 public class MyScreen extends Activity {
2
3     @Override
4     protected void onCreate(Bundle
5         savedInstanceState) {
6         super.onCreate(
7             savedInstanceState);
8         //Mise en place des vues ici
9     }
10
11     @Override

```

```

10     protected void onStart() {
11         super.onStart() ;
12         new MyRPC(this).execute(arg1,
13             arg2, arg3, ...); //
14             Ordonne l'appel asynchrone
15             avec des arguments de type X
16     }
17
18     public void populate(Z data) {
19         //Mise à jour des vues avec data
20         de type Z
21     }

```

Le code de l'appel distant sera quant à lui écrit dans votre classe générique *MyRPC*, paramétrée par trois types de votre choix :

- X : le type des données d'entrées pour réaliser l'appel;
- Y : le type de l'unité de progression (le plus souvent un entier);
- Z : le type du résultat de retour de l'appel.

Une *AsyncTask* possède les trois méthodes importantes *onPreExecute()*, *doInBackground()* et *onPostExecute()*, qui sont invoquées automatiquement dans cet ordre par le système Android dès lors que vous invoquez la méthode *execute()*. Il faut alors comprendre le cheminement suivant : ce que vous passez en entrée de *execute()* se retrouve en entrée de *doInBackground()* où une requête HTTP sera forgée via un client, et le retour de cette dernière est implicitement réinjecté en entrée de *onPostExecute()* pour lui appliquer un post-traitement. En l'occurrence, notre patron de conception stipule de faire remonter cette valeur jusqu'à l'écran via *populate()*.

```

1 public class MyRPC extends AsyncTask<X,
2     Y, Z> {
3
4     private volatile MyScreen screen;
5     //référence à l'écran
6     private HttpClient client;
7     //référence au client HTTP
8
9     public MyRPC(MyScreen s) {
10         this.screen = s ;
11         this.client = new
12             DefaultHttpClient();
13     }
14
15     @Override
16     protected void onPreExecute() {
17         //prétraitement de l'appel
18     }
19
20     @Override
21     protected Z doInBackground(X...
22         params) {
23         //Appel de procédure distante
24         via HTTP (traitement long)
25         Z obj = this.client.execute(
26             url_with_params);
27         return obj; //Objet de type Z
28         automatiquement réinjecté en
29         entrée de onPostExecute()
30     }

```

```

22
23     @Override
24     protected void onPostExecute(Z
25         result) {
26         //post-traitement de l'appel
27         this.screen.populate(result); //
28         //callback
29     }

```



Lorsque la « configuration » du périphérique change (vous pivotez votre appareil, vous modifiez la langue de l'OS...), l'instance de l'« Activity » est automatiquement détruite et une nouvelle est recréée. Le champ screen est donc déclaré volatile pour éviter d'utiliser une référence périmée qui aurait été mise en cache.

3 Cas d'études

Afin d'expérimenter le patron ci-dessus, considérons trois cas d'études différents. Chaque projet étant unique, les possibilités sont infinies, mais elles peuvent toutefois être catégorisées en trois grandes familles :

- la procédure appelée retourne un flux d'octets (image, sons, binaires en tout genre...);
- la procédure appelée retourne un document (texte, JSON, XML et ses dérivés...);
- la procédure appelée ne retourne rien.

Dans un souci de clarté, dans les portions de code qui vont suivre, le traitement des exceptions est réduit au minimum syndical. Un tutoriel entier est consacré à cet aspect : [lien 78](#). De même, les contrôles des codes de retour HTTP du serveur (200, 404, 403...) sont volontairement omis.

3.1 Une image

Imaginons un programme serveur qui renvoie une image lorsqu'il est appelé. Ce peut être une image stockée physiquement sur un serveur comme celui-ci ([lien 79](#)), ou — plus intéressant — une image générée à la volée en fonction de paramètres comme ceci : [lien 80](#).



Cette dernière URL doit être vue comme la signature de la fonction `Byte chart(String cht, String chs, String chl)` située sur la machine distante `http://chart.apis.google.com/`.

```

1 public class ImageScreen extends
2     Activity {
3     private ImageView viewer;
4
5     @Override
6     protected void onCreate(Bundle
7         savedInstanceState) {
8         super.onCreate(
9             savedInstanceState);
10        setContentView(R.layout.
11            image_activity);
12        this.viewer = (ImageView)
13            findViewById(R.id.imageView1
14            );

```

```

10
11
12     @Override
13     protected void onStart() {
14         super.onStart();
15         new DownloadImage(this).execute(
16             "qr", "200x200", "Coucou");
17     }
18     public void populate(Bitmap data) {
19         this.viewer.setImageBitmap(data)
20         ;
21     }
22 }

```

```

1 public class DownloadImage extends
2     AsyncTask<String, Void, Bitmap> {
3     private static final String BASE_URL
4         = "http://chart.apis.google.com
5         /chart";
6     private volatile ImageScreen screen;
7     private HttpClient client;
8     private ProgressDialog progress;
9
10    public DownloadImage(ImageScreen s)
11    {
12        this.screen = s;
13        this.client = new
14            DefaultHttpClient();
15        this.progress = new
16            ProgressDialog(this.screen);
17    }
18
19    @Override
20    protected void onPreExecute() {
21        this.progress.setTitle("Veuillez
22            patienter");
23        this.progress.setMessage("Récupé
24            ration de l'image en cours
25            ...");
26        this.progress.setProgressStyle(
27            ProgressDialog.STYLE_SPINNER
28            );
29        this.progress.show();

```

```

30     String url = String.format("
        %s?cht=%s&chs=%s&chl=%s"
        , BASE_URL, params[0],
        params[1], params[2]);
31     HttpGet request = new
        HttpGet(url);
32     HttpResponse response = this
        .client.execute(request)
        ;
33     HttpEntity entity = response
        .getEntity();
34     byte[] bytes = EntityUtils.
        toByteArray(entity);
35     dummyImage = BitmapFactory.
        decodeByteArray(bytes, 0
        , bytes.length);
36
37     } catch (Exception e) { Log.e("
        RPC", "Exception levée", e);
        }
38
39     return dummyImage;
40 }
41
42 @Override
43 protected void onPostExecute(Bitmap
        result) {
44
45     if(this.progress.isShowing())
46         this.progress.dismiss();
47     this.screen.populate(result);
48
49 }

```

3.2 Du texte brut

Imaginons maintenant un programme serveur qui renvoie du texte (poèmes, définitions, blagues...). En guise d'illustration, j'ai choisi l'URL qui renvoie du texte aléatoirement. Bien entendu, les données auraient tout aussi bien pu être structurées en JSON ou en XML. Quel que soit le format, il vous faudra « parser » les données reçues. L'API Android contient tout ce qu'il faut pour cela, et notamment la classe Scanner quand il s'agit de texte libre comme c'est le cas ici.

```

1 public class LoremScreen extends
        ListActivity {
2
3     private ArrayAdapter<String> adapter
        ;
4
5     @Override
6     protected void onCreate(Bundle
        savedInstanceState){
7         super.onCreate(
            savedInstanceState);
8         setContentView(R.layout.
            lorem_activity);
9         this.adapter = new ArrayAdapter<
            String>(this, android.R.
            layout.simple_list_item_1);
10        this.setAdapter(adapter);
11    }
12
13    @Override
14    protected void onStart() {
15        super.onStart();

```

```

16        new DownloadLorem(this).execute
            ();
17    }
18
19    public void populate(ArrayList<
        String> data) {
20        this.adapter.clear();
21        this.adapter.addAll(data);
22        this.adapter.
            notifyDataSetChanged();
23    }
24
25 }

```

```

1 public class DownloadLorem extends
        AsyncTask<Void, Void, ArrayList<
        String>> {
2
3     private static final String BASE_URL
        = "http://loripsum.net/api/
        plaintext/short/20/";
4     private LoremScreen screen;
5     private HttpClient client;
6     private ProgressDialog progress;
7
8     public DownloadLorem(LoremScreen s)
        {
9         this.screen = s;
10        this.client = new
            DefaultHttpClient();
11        this.progress = new
            ProgressDialog(this.screen);
12    }
13
14    @Override
15    protected void onPreExecute() {
16        progress.setTitle("Veuillez
            patienter");
17        progress.setMessage("Récupé
            ration des données en cours
            ...");
18        progress.setProgressStyle(
            ProgressDialog.STYLE_SPINNER
            );
19        progress.show();
20    }
21
22    @Override
23    protected ArrayList<String>
        doInBackground(Void... params) {
24
25        InputStream content = null;
26        ArrayList<String> dummyTexts =
            new ArrayList<String>();
27
28        try {
29            HttpResponse response = this
                .client.execute(new
                HttpGet(BASE_URL));
30            content = response.getEntity
                ().getContent();
31            Scanner scanner = new
                Scanner(new
                InputStreamReader(
                content));
32            scanner.useDelimiter("\\s*\n
                ");
33            while (scanner.hasNext()) {
                dummyTexts.add(scanner.
                next().substring(0,30).
                concat("...")); }
34
35        } catch (Exception e) { Log.e("
            RPC", "Exception levée", e);

```

```

36     }
37     return dummyTexts;
38 }
39
40 @Override
41 protected void onPostExecute(
42     ArrayList<String> result) {
43     if(progress.isShowing())
44         progress.dismiss();
45     this.screen.populate(result);
46 }
47 }

```

3.3 Rien du tout

Enfin, imaginons un programme serveur codé en PHP et qui stocke anonymement les démarrages et arrêts de votre application à des fins statistiques. Pour en avertir le serveur, l'URL de la procédure serait de la forme : . Comme il n'y a pas de retour de l'appel distant, le callback populate() du patron de conception est inutile.

```

1 public class LogScreen extends Activity
2 {
3     private TelephonyManager tm;
4
5     @Override
6     protected void onCreate(Bundle
7         savedInstanceState) {
8         super.onCreate(
9             savedInstanceState);
10        setContentView(R.layout.
11            log_activity);
12
13        tm = (TelephonyManager) this.
14            getSystemService(Context.
15                TELEPHONY_SERVICE);
16        new NotifyLogger().execute(tm.
17            getDeviceId(), "CONNECT");
18    }
19 }

```

```

13
14 @Override
15 protected void onDestroy() {
16     super.onDestroy();
17     new NotifyLogger().execute(tm.
18         getDeviceId(), "DISCONNECT")
19     ;
20 }

```

```

1 public class NotifyLogger extends
2     AsyncTask<String, Void, Void> {
3
4     private static final Object BASE_URL
5         = "http://my.company.com/remote
6         /logUserConnexion.php";
7
8     private HttpClient client;
9
10    public NotifyLogger() {
11        this.client = new
12            DefaultHttpClient();
13    }
14
15    @Override
16    protected Void doInBackground(String
17        ... params) {
18        try {
19            String url = String.format("
20                %s?id=%s&event=%s",
21                BASE_URL, params[0],
22                params[1]);
23            HttpGet request = new
24                HttpGet(url);
25            this.client.execute(request)
26            ;
27        } catch (Exception e) { Log.e("
28            RPC","Exception levée", e);
29        }
30
31        return null;
32    }
33 }

```

4 Code source

La totalité du code source utilisé pour ce tutoriel est téléchargeable sous forme d'une archive de projet Eclipse : [lien 83](#).

Retrouvez l'article d'*Olivier Le Goer* en ligne : [lien 84](#)



Programmation

Les derniers tutoriels et articles

Le débogage d'une application : méthodes et exercices

Découvrez comment l'ordinateur perçoit votre programme

Vous êtes face à un bogue et vous ne savez pas où il se trouve. Les `printf/cout/println` (ou autres fonctions affichant du texte) ne mènent à rien de concret et ne vous aident pas. On vous dit d'utiliser un « débogueur ». Vous en avez un, mais vous ne l'avez jamais utilisé et vous ne savez pas quoi faire. Cet article est pour vous !

1 Introduction

Lorsque nous programmons, nous créons des bogues. Il existe des méthodes pour les éviter ou pour les débusquer plus rapidement (assertions, tests unitaires, avertissement du compilateur...), mais cela ne suffit généralement pas. Vous êtes maintenant face à un bogue, vous avez un débogueur, mais vous ne savez pas comment découvrir l'origine de celui-ci. Vous êtes bloqué à relire votre code et vous n'avancez plus. Ce n'est pas grave, ce tutoriel est là pour guider vos premiers pas vers la solution.



Ce tutoriel vise à vous apprendre la réflexion à adopter lorsque vous vous retrouvez face à un bogue. En effet, les bogues sont uniques pour chaque programme et donc, au lieu de vous apprendre à résoudre des cas d'école, ce tutoriel vous apprendra à vous adapter et à analyser la situation pour localiser et supprimer tous les bogues.

chine peut ne pas interpréter votre code exactement comme vous l'avez imaginé. Celle-ci est très « naïve » et effectue exactement ce qui est écrit et non ce que vous avez pu imaginer. Le fait de lui faire comprendre exactement ce que vous voulez, de lui ordonner exactement ce que vous souhaitez lui faire faire fait partie de la difficulté de la programmation. En résumé, il faut se mettre à la place de la machine. Comme celle-ci n'est pas aussi évoluée que nous, il arrive un moment où nous ne pouvons pas juste deviner sa compréhension de notre code. On peut s'aider des fonctions d'affichage pour forcer la machine à nous indiquer la valeur des variables et une partie des opérations qu'elle effectue, mais cela ne suffit que très rarement. C'est là que le débogueur entre en jeu. Celui-ci permet de suivre exactement ce que fait la machine, de l'observer à la loupe afin de mieux comprendre son interprétation du code. Ainsi, en voyant ce qu'elle fait, vous pourrez découvrir à quel endroit la machine n'a pas agi comme vous le souhaitez et modifier le programme afin que son exécution par la machine corresponde à vos souhaits. C'est en supprimant ces différences entre ce que vous voulez faire faire à la machine et ce qu'elle comprend de vos instructions que vous allez supprimer les bogues. Le débogueur est l'outil pour analyser, observer, vérifier le comportement de la machine lors du traitement de votre programme.

Et vous allez le voir, sa compréhension de votre programme est souvent éloignée de ce que l'on imagine.

1.1 Qu'est-ce que le débogage ?

Le débogage (de l'anglais « to debug ») indique l'action de retirer un bogue.

Un bogue, c'est lorsque vous avez programmé quelque chose, afin d'obtenir un résultat précis, mais que vous obtenez autre chose. Bien que vous ayez écrit le code pour réaliser une tâche, l'ordinateur ne le comprend pas et fait autre chose. En effet, la ma-

2 Les débogueurs

2.1 Introduction

Les débogueurs sont des outils très puissants. Lorsque vous lancez votre programme au sein du débogueur, celui-ci suit le comportement de votre application et si un crash se produit, il s'arrête sur la ligne provoquant l'erreur. À partir de là, votre programme n'est pas réellement arrêté, il est en pause, mais ne pourra pas continuer. Le débogueur vous permet d'analyser son état, de voir les valeurs des variables, d'afficher la liste des fonctions appelées et ainsi de comprendre comment le programme en est arrivé là.

Si vous croyez que le débogueur ne sert qu'en cas de crash, c'est que vous ne connaissez pas toute sa puissance. En effet, vous pouvez indiquer à votre débogueur de s'arrêter à une ligne précise de l'application afin d'analyser l'état de celle-ci à cet endroit précis du code, ou même de l'exécuter ligne par ligne pour comprendre comment la machine interprète votre code.

Comme pour tout outil, il faut lire son mode d'emploi. Je vais fournir les informations de manière généraliste, afin que vous puissiez utiliser n'importe quel débogueur.

En complément, vous pouvez aussi lire les tutoriels suivants :

- tutoriel sur DDD (une interface graphique pour GDB), de Hiko Sejura : [lien 85](#) ;
- tutoriel sur Microsoft Visual Studio de Laurent Gomila : [lien 86](#).



N'hésitez pas à tester les manipulations expliquées dans ce tutoriel. Il est plus simple d'apprendre tout en pratiquant.

2.2 Les débogueurs

Chaque EDI intègre un débogueur et cela, quel que soit le langage. Un débogueur peut être utilisable en ligne de commande ou à travers une interface graphique, mais dans tous les cas, il propose toujours les mêmes fonctionnalités.

Ainsi, ce tutoriel ne va pas nécessairement se reposer sur tel ou tel débogueur, mais vous expliquer de manière globale ce qu'un débogueur vous propose et comment l'utiliser pour comprendre le déroulement de votre programme.

2.3 Fonctionnalités

Un débogueur, qu'il soit en ligne de commande ou possédant une interface proposera (entre autres) les fonctionnalités suivantes :

- la possibilité de placer des points d'arrêt ;

- la possibilité de voir les valeurs des variables ;
- la possibilité d'afficher la pile d'appels ;
- la possibilité d'exécuter le programme pas à pas.

2.4 Définitions

2.4.a Les points d'arrêt

Un point d'arrêt permet d'indiquer au débogueur de s'arrêter à une ligne spécifique dans le programme. En effet, pendant l'exécution du programme dans le débogueur, le débogueur s'arrêtera avant d'exécuter la ligne où le point d'arrêt est placé. Une fois le programme arrêté, il vous est possible d'afficher les valeurs des variables, ou de continuer l'exécution.



Généralement, un point d'arrêt est représenté par un point rouge dans la colonne à gauche du code (à côté des numéros de ligne).

2.4.b Affichage des valeurs des variables

Lorsque le programme est arrêté au sein du débogueur (à l'aide d'un point d'arrêt, par exemple), il est possible d'afficher les variables utilisées par le programme.

Généralement, il suffit de laisser le curseur sur la variable, pour connaître sa valeur, ou en effectuant un clic droit sur la variable pour ajouter un moniteur (*watch*).

2.4.c Pile d'appels

La pile d'appels est un mécanisme permettant au programme de revenir à la fonction appelante, lorsque l'exécution d'une fonction est finie. Pour cela, à l'appel d'une fonction, celle-ci est enregistrée dans la pile d'appels. Une fois que la fonction est finie, le programme doit retourner dans la fonction appelante pour continuer l'exécution du programme. Pour ce faire, le processeur regarde la dernière fonction placée dans la pile d'appels et y retourne tout en l'enlevant de la pile.

La pile d'appels peut être affichée à l'aide de votre débogueur afin de voir quelles sont les fonctions qui ont été appelées pour arriver à ce point du programme.

La pile d'appels est aussi très intéressante, car lorsqu'un crash est détecté par le débogueur, il survient généralement dans les fonctions de la bibliothèque que vous utilisez (comme la bibliothèque standard du C) et vous devez remonter la pile d'appels afin de savoir par quel chemin votre programme est passé pour arriver à ce crash.



Même si votre débogueur stoppe dans les fonctions d'une bibliothèque, il ne faut pas immédiatement rejeter la faute sur la bibliothèque. En effet, si un crash survient, cela sera sûrement dû à votre mauvaise utilisation de la fonction de la bibliothèque.

Notez qu'une bibliothèque est un code utilisé par de nombreuses personnes. Il a été testé et mis à l'épreuve, vous pouvez donc le considérer comme sain et sans bogue.

2.4.d Exécution pas à pas (contrôle de l'exécution)

Une fois que le programme est mis en pause dans le débogueur, il y a plusieurs façons de continuer son exécution. Vous pouvez demander au débogueur de continuer l'exécution et il s'arrêtera au prochain point d'arrêt ou finira l'exécution du programme. Vous pouvez aussi demander d'exécuter la ligne actuelle et le débogueur s'arrêtera à la ligne suivante (appelée exécution pas à pas). Cela est très

pratique pour vérifier que les valeurs des variables sont cohérentes et correspondent à ce que vous souhaitez. Vous pouvez aussi demander au débogueur d'exécuter la ligne actuelle, tout en entrant dans les sous-fonctions. Le comportement est proche du cas précédent, mais permet aussi de déboguer les sous-fonctions.



Vous pouvez entrer dans les sous-fonctions des bibliothèques, mais généralement cela ne mène à rien, car soit le code n'est pas accessible, soit vous pouvez le considérer comme exempt de bogue.

Toutefois, lorsque vous avez accès à son code, cela peut être pratique pour comprendre le fonctionnement de la bibliothèque (en plus de sa documentation).



Lorsque le débogueur s'arrête à la suite d'un crash, il n'est pas possible de continuer l'exécution du programme. Si vous essayez, le programme sera complètement arrêté.

3 Utilisation d'un débogueur

Tous les débogueurs proposent une série de fonctionnalités communes permettant d'analyser le comportement d'un programme. Dans ce chapitre, je vais décrire comment utiliser les débogueurs les plus courants : GDB, Code : :Blocks, Microsoft Visual Studio, Eclipse. Vous pouvez utiliser un autre débogueur et vous remarquerez très rapidement que son utilisation est similaire à celle vue ici. Vous remarquerez que j'ai aussi intégré Eclipse. Quel que soit le langage de programmation, les fonctionnalités proposées par le débogueur restent les mêmes. Finalement, que vous déboguez un programme C/C++/Java ou autre, le procédé reste le même et la méthode est identique.

3.1 GDB, un débogueur en ligne de commande


GDB est une abréviation pour GNU Debugger. Il est disponible pour une multitude de systèmes.

GDB est un débogueur fonctionnant dans la console. Il est principalement utilisé sous Linux et peut être quelque peu austère pour les débutants.

Toutefois, ce logiciel dépanne bon nombre de programmeurs et malgré une utilisation en ligne de commande, son efficacité n'en est pas réduite.

Nombreux sont les éditeurs de code intégrant une surcouche à GDB. En effet, Code : :Blocks et Qt Creator peuvent utiliser GDB en arrière-plan pour vous proposer un débogage graphique. Sachez aussi qu'il existe DDD, une interface graphique pour GDB, sous Linux. Ce dernier peut vous permettre d'apprendre à utiliser GDB, car il indique toutes les commandes passées au débogueur.

3.2 Code : :Blocks/Microsoft Visual Studio/Eclipse




























Les logiciels Code : :Blocks/Microsoft Visual Studio/Eclipse sont des éditeurs de développement intégrés (EDI) et proposent donc un débogueur intégré. Directement dans l'interface de votre code, vous accédez aux fonctionnalités du débogueur, après avoir lancé le programme dans celui-ci (généralement à l'aide du menu : *Debug/Start* ou l'icône en forme de curseur de lecture : ).

3.3 Débogage

3.3.a Découverte des fonctionnalités

3.3.a.a Commandes et icônes

Voici les commandes et boutons disponibles dans chacun des débogueurs :

Action	GDB (commandes)	Code :: Blocks	Microsoft Visual Studio (<=2010)	Microsoft Visual Studio 2013	Eclipse	Qt Creator
Démarrer ou continuer l'exécution du code jusqu'à la fin, ou le prochain point d'arrêt	run/continue					
Exécuter une ligne (sans entrer dans la fonction)	next					
Exécuter une ligne (en entrant dans la fonction)	step					
Continuer le programme jusqu'à sortir de la fonction actuelle	finish					
Stopper le débogage (et le programme débogué)	stop					
Pour afficher la pile d'appels, les moniteurs...	backtrace display					
Placer un point d'arrêt	break fichier :numéro_de_ligne	Un clic à côté du numéro de la ligne permet de placer un point d'arrêt (cela rajoute un petit cercle pour indiquer la présence du point d'arrêt).				



Les EDI proposent des dispositions de fenêtres propres au débogage (intégrant les fenêtres des moniteurs, de pile d'appels et autres fenêtres) qui sont sauvegardées à la fin du débogage.

3.3.a.b Point d'arrêts

Les points d'arrêts sont une arme redoutable face aux bogues. Ceux-ci permettent d'arrêter le programme n'importe où. Il suffit de spécifier la ligne sur laquelle le débogueur doit s'arrêter et il le fera juste avant l'exécution de celle-ci.

Action	GDB (commandes)	Code : : Blocks	Microsoft Visual Studio (<=2010)	Microsoft Visual Studio 2013	Eclipse	Qt Creator
Placer un point d'arrêt	break fichier :numéro_de_ligne	Un clic à côté du numéro de la ligne permet de placer un point d'arrêt (cela rajoute un petit cercle pour indiquer la présence du point d'arrêt).				

Une fois que le programme a atteint le point d'arrêt, le débogueur le met en pause. À partir de là, vous pouvez soit reprendre l'exécution du programme, soit afficher les valeurs des variables ou de la liste d'appels.

3.3.a.c Contrôle de l'exécution

Les commandes suivantes permettent de contrôler le code exécuté. En effet, lorsque votre programme s'est arrêté à un point d'arrêt, vous avez plusieurs choix pour avancer dans le programme :

Action	GDB (commandes)	Code : : Blocks	Microsoft Visual Studio (<=2010)	Microsoft Visual Studio 2013	Eclipse	Qt Creator
Exécuter une ligne (sans entrer dans la fonction)	next					
Exécuter une ligne (en entrant dans la fonction)	step					
Continuer le programme jusqu'à sortir de la fonction actuelle	finish					

Voici où chaque commande emmènera le programme après son exécution. Le programme est actuellement arrêté par un point d'arrêt sur la ligne 10 (indiquée par la flèche) :

```

1  #include <stdio.h>
2
3  void bar()
4  {
5      printf("Bonjour monde\n");
6  }
7
8  void foo()
9  {
10     bar();
11 }
12
13 int main(int argc, char** argv)
14 {
15     foo();
16     return 0;
17 }
18

```

En détail, il faut comprendre que :

- **next** exécutera la fonction bar(), comme si cela était une unique instruction (« Bonjour monde » est affiché) ;
- **step** exécutera l'appel de la fonction bar() et s'arrêtera avant le printf() (« Bonjour monde » n'est pas affiché) ;
- **finish** sortira de la fonction foo() en exécutant le reste de la fonction (donc, la fonction bar()). Le débogueur s'arrêtera avant le return 0 (« Bonjour monde » est affiché et le programme ne quitte pas le main).

Avec ces commandes, vous pouvez précisément contrôler l'avancement du programme et ainsi vérifier chacun des calculs effectués par celui-ci.

3.3.a.d Affichage de valeurs

Notre programme est en pause dans le débogueur. Celui-ci peut afficher le contenu de chacune des variables accessibles à l'endroit où le programme est.



Action	GDB (commandes)	Code : : Blocks	Microsoft Visual Studio (<=2010)	Microsoft Visual Studio 2013	Eclipse	Qt Creator
Afficher le contenu d'une variable	print nom_de_la_variable	Souvent, il suffit de laisser le curseur quelques instants sur la variable. Sinon, vous pouvez faire un clic droit et cliquer sur « ajouter moniteur »/« ajouter à la fenêtre des observateurs » (« Add Watch »/« Watch... »).				



Seules les variables actuellement visibles par le programme peuvent être affichées. En effet, la règle de la portée des variables s'applique, car les variables hors de portée n'existent pas et donc ne peuvent pas être affichées.

3.3.a.e Pile d'appels

La pile d'appels correspond à la liste des fonctions qui ont été appelées pour arriver à un point précis du programme. Pour faire simple, c'est le chemin qu'a pris le programme.

Action	GDB (commandes)	Code : : Blocks	Microsoft Visual Studio (<=2010)	Microsoft Visual Studio 2013	Eclipse	Qt Creator
Pour afficher la pile d'appels, les moniteurs...	backtrace display					



En anglais la pile d'appels est appelée « backtrace » ou encore « call stack ».


Vous pouvez vous déplacer dans la liste d'appels (et afficher le code d'appel à telle ou telle fonction) en cliquant sur les lignes affichées ou avec les commandes GDB up/down (pour remonter ou redescendre dans la pile).

```

Call stack
-----
Nr | Address | Function
---|---|---
0 0x7ffff72dfa75 * _GI_raise(sig=<value optimised out>)
1 0x7ffff72e35c0 * _GI_abort()
2 0x7ffff73194fb __libc_message(do_abort=<value optimised out>, fmt=<
3 ( 0x00007ffff73235b6 in malloc_printerr(action=3, str=
4 0x7ffff7329e83 * _GI___libc_free(mem=<value optimised out>)
5 0x400610 main()
  
```

Pile d'appels de Code : :Blocks

Généralement, la pile d'appels affiche le nom des fonctions appelées (avec leurs arguments et, si possible, la valeur de ces arguments), l'adresse mémoire de l'appel, le nom du fichier et la ligne où l'appel a lieu. De plus, il arrive souvent que les appels indiqués présentent des fonctions d'une bibliothèque, vous devez vous concentrer sur l'appel le plus profond de **votre code**. Dans l'exemple ci-dessus, `raise/abort/___libc_message/malloc_printerr/___GI___libc_free` sont des fonctions de la bibliothèque standard (appelées par `free()`) et ne sont pas intéressantes. La ligne qui nous intéresse donc est celle du `main()`, car c'est l'origine du crash dans notre code.



Il faut comprendre que la bibliothèque standard ou les autres bibliothèques ne sont pas boguées. Elles crashent suite à une mauvaise utilisation effectuée par le programmeur. Dans l'exemple ci-dessus, `free()` crashe, car le pointeur passé à `free()` est invalide.

3.3.b Utilisation du débogueur

Nous avons vu les fonctionnalités de base du débogueur et nous savons maintenant comment les activer. Tout cela est bien beau, mais il faut désormais comprendre ce qu'est le débogage et l'attitude qu'il faut avoir face à un bogue.

3.3.b.a Qu'est-ce qu'un bogue ?

Généralement, un bogue est un comportement inattendu d'un programme. Par exemple, vous souhaitez que votre programme fasse une opération précise : une addition. Suivant les entrées que vous lui donnez, vous espérez avoir le résultat qui suit les règles mathématiques. Toute déviation de ce résultat sera considérée comme un bogue. En effet, vous ne voulez pas que le programme réponde 3 lorsque vous lui donnez en entrée 1 et 1. Mais ce n'est pas tout. Certains bogues provoquent un crash du programme, ce qui fait qu'ils sont très visibles et aussi très gênants. Il existe d'autres bogues, totalement invisibles, qui ne crasheront pas le programme, et qui ne donneront pas nécessairement de mauvais résultats. En effet, cela peut être simplement une mauvaise interaction avec le système, ou encore une occupation en mémoire jugée trop importante.

En résumé, un bogue est un comportement non voulu du programme.

3.3.b.b Comment corriger un bogue ?

Pour corriger un bogue, il faut comprendre la raison de sa présence. Tant que vous ne comprenez pas son origine et pourquoi il se produit, vous serez incapable de mettre en place une correction totalement

efficace.

Pour comprendre la raison d'un bogue, vous avez plusieurs fois :

- relu le code et vous voyez une erreur dans celui-ci. Cela ne fonctionne que pour les petites erreurs « évidentes » ou grâce à l'expérience, mais sera inefficace dans les cas où des structures de données ou des algorithmes un peu plus évolués sont intégrés au code ;
- ajouté des `cout/print` pour afficher la valeur des variables. Dans un sens, c'est ce que fait le débogueur. Certes, c'est rapide. Malheureusement, le fait d'ajouter un appel de fonction peut faire disparaître la conséquence du bogue (mais pas le bogue en lui-même) ;
- essayé de déduire l'erreur, à l'aide d'une multitude de variables d'entrées. Si le bogue a un comportement « logique » et « régulier », alors vous pourrez le deviner suivant les différents résultats du programme. Toutefois, c'est inefficace dans un cas où le bogue est lié à la mémoire ;
- il ne reste plus que le débogueur. Celui-ci vous permettra d'analyser aussi bien les valeurs des variables que la mémoire, mais aussi de suivre le programme pas à pas et de valider son comportement.

Une fois que vous avez trouvé un bogue, il suffit de le corriger dans le code source. Si vous avez compris ce qui se passait et pourquoi le bogue apparaissait, la correction vous sera évidente.



Lors d'un crash, le débogueur aura un avantage indéniable. En effet, le débogueur s'arrête à la ligne provoquant le crash. C'est bien plus efficace que de placer une multitude de print/-cout afin de voir le dernier qui s'affiche. D'autant plus que les fonctions d'affichage sont mises en tampon et peuvent ne pas être affichées immédiatement (et donc ne jamais s'afficher lors d'un crash).

3.3.b.c Débusquer un bogue avec le débogueur : un jeu d'enquêteur

Supprimer un bogue est très semblable à une enquête policière. Vous lancez votre programme et celui-ci produit un bogue. Vos premiers indices sont le mauvais comportement du programme. Pourquoi produit-il ces mauvais résultats ?

- Un crash ? Pourquoi le programme crashe, ou plutôt, où le programme crashe ? En lançant le programme dans un débogueur, ce dernier s'arrêtera à la ligne du crash. Parfait pour connaître où le programme crashe. Si le débogueur s'arrête dans les fonctions d'une bibliothèque, regardez la **pile d'appels** pour voir

quelle est la dernière fonction de votre programme qui a été appelée. Bien souvent, ce crash est dû à un pointeur invalide (ou nul) ou de mauvaises valeurs de variables passées à une fonction.

- Un mauvais résultat ? Réfléchissez à la façon dont vous produisez ces résultats. Quelles sont les fonctions à l'origine de ce résultat. Placez un **point d'arrêt** au début des fonctions suspectées et vérifiez que chacune d'elles fait correctement son travail. Pour cela, n'hésitez pas à exécuter les fonctions **pas à pas** pour vérifier que les valeurs des variables restent correctes (en les **affichant**).
- Une condition ne s'exécute pas alors que, dans un tel cas, elle le devrait. Placez un **point d'arrêt** devant la condition. Vérifiez les valeurs des variables dans le test, en les **affichant**.



N'hésitez pas à utiliser le débogueur et à analyser le comportement de votre programme avec celui-ci. En effet, la vision que vous avez de votre code peut être très loin de l'utilisation que l'ordinateur en fera. Très souvent, il ne fait pas ce que vous pensiez.

4 Cas pratique

Nous allons maintenant étudier et comprendre plusieurs cas d'erreur courants afin que vous sachiez quoi faire lorsque vous allez avoir des bogues. Malheureusement, malgré ma volonté de vouloir faire un article générique, mes exemples seront dépendants d'un langage (ici, le C). Toutefois, la façon de penser pour retrouver une erreur est la même pour tous les langages.

4.1 Double affichage d'un menu

Le code suivant contient un bogue. Il affiche deux fois le menu, sans raison !

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 unsigned short menu()
6 {
7     char saisie[100];
8     unsigned short choix;
9     while(1)
10    {
11        printf("\t*****
        Bienvenue Au Grand Bazar ***
        *****\n\n");
12        printf("0 :Quitter\n");
13        printf("\nVotre choix : ");
14        fflush(stdout);
15    }

```

```

16    fgets(saisie,100,stdin);
17    if (sscanf(saisie, "%hu", &choix
18        ) == 1 && choix == 0) break;
19    printf("Choix incorrect -
20        Recommencez !!!\n");
21    }
22    return choix;
23 }
24 int main()
25 {
26    //Mise en place d'un systeme de mot
27    de passe
28    int motpasse =77;
29    int code;
30
31    do
32    {
33        printf("veuillez saisir le code
34            : ");
35        scanf("%d",&code);
36
37        if(code==motpasse)
38        {
39            menu();
40        }
41        else
42        {
43            printf("Erreur de code !!!")
44            ;
45        }
46    }
47    while (code !=motpasse);

```

```
44     return 0;
45 }
46 }
```

Si vous l'exécutez et que vous entrez 77 (le code valide), vous obtenez :

```
1  veuillez saisir le code : 77
2  ***** Bienvenue Au
   Grand Bazar *****
3
4
5  0 : Quitter
6
7  Votre choix : Choix incorrect -
   Recommencez !!!
8  ***** Bienvenue Au
   Grand Bazar *****
9
10
11 0 : Quitter
12
13 Votre choix :
```

Pourquoi ce code affiche-t-il deux fois le menu ? Le débogueur va être là pour mettre en évidence un comportement ne semblant pas avoir été prévu.

Ce que nous voyons, c'est un double affichage du menu. Mettons donc un point d'arrêt là où commence l'affichage du menu : à la ligne 9, `while(1)`. Si nous lançons le programme (dans le débogueur), il va nous demander le mot de passe. Si nous mettons un mot de passe faux, il boucle. Si nous mettons un mot de passe juste, il va s'arrêter sur le point d'arrêt. Jusque-là, nous pouvons présumer que le code pour le mot de passe est juste et sans bogue. Nous sommes donc à la ligne 9. Le menu n'est pas encore affiché. Passons les lignes 10 à 16, avec la commande **step**. Comme nous exécutons le programme ligne par ligne, nous voyons les lignes s'afficher à chaque **step** effectué. Enfin, nous arrivons à la ligne 16. Lorsque cette ligne va s'exécuter, elle doit demander à l'utilisateur d'entrer quelque chose. Soit, exécutons la ligne avec **step**. Celle-ci passe à la ligne suivante,

sans même attendre que nous rentrions quoi que ce soit ! Si nous continuons le programme, on voit bien qu'il affiche une seconde fois le menu et que `fgets()` se bloque comme attendu. `fgets()` ([lien 87](#)), dans notre cas, lit sur l'entrée standard. Habituellement, la fonction se bloque en attendant que l'utilisateur rentre son choix. `fgets()` provenant de la bibliothèque standard n'est pas boguée. Mais alors, que contient donc la variable `saisie`, variable utilisée pour récupérer ce que `fgets()` a lu ? Le débogueur peut nous le dire. Affichons donc cette variable :

```
1  saisie "\n\000\377\367..."
```

Chez vous, il peut y avoir d'autres choses qui suivent, mais les deux premiers caractères de la chaîne de caractères `saisie` sont toujours les mêmes : `\n` et `\000`. `\n` indique un saut de ligne. `\000` c'est équivalent à `\0` ou, plus précisément, la fin d'une chaîne de caractères. La question est donc : d'où vient ce `\n` ? Simplement, il vient de la saisie du mot de passe faite précédemment. En effet, le `scanf()` pour le mot de passe ne lit que le nombre et pas le `\n` qui est inséré par l'appui sur la touche entrée. Du coup, le « `\n` » reste dans le tampon d'entrée et est par la suite récupéré par le `fgets()`. La fonction pense que c'est une entrée utilisateur et donc continue le programme. Et comme le choix est invalide, le programme affichera une seconde fois le menu.

Pour corriger le problème maintenant repéré, il suffit de vider le tampon d'entrée : [lien 88](#).

Ici, le débogueur nous a permis de mieux comprendre comment le programme fonctionnait. Grâce à l'exécution du programme pas à pas, nous avons pu voir un comportement inattendu (le `fgets` qui ne bloque pas), mais aussi nous avons pu comprendre la raison (affichage de la variable `saisie`). Une fois le bogue débusqué, il suffit de le corriger et tout va mieux.

Source du problème : [lien 89](#)

5 Erreur de segmentation (segmentation fault)

L'erreur de segmentation est une erreur très répandue, mais loin d'être difficile à corriger.

Tout d'abord, il faut comprendre que l'erreur de segmentation signifie que votre programme a été arrêté par le système d'exploitation suite à une opération invalide. L'opération invalide la plus courante est un accès mémoire interdit. En effet, votre programme se voit attribuer une partie de la mémoire. Si vous tentez un accès (lecture ou écriture) en dehors de la zone que le système vous a attribuée, celui-ci va vous stopper. Malheureusement, ce n'est pas toujours vrai. Cela est même « aléatoire » (comprendre : non prévisible, car dépend de l'agencement actuel de la mémoire, donc de l'utilisation passée de la mémoire par le reste du système).

Lorsque vous avez une erreur de segmentation,

la première chose à faire est de voir si l'erreur est reproductible. Une erreur reproductible est plus facile à étudier. De plus, si vous lancez le programme défectueux dans un débogueur, ce dernier s'arrêtera en vous affichant la ligne fautive. Une fois la ligne identifiée, il suffit généralement d'afficher les valeurs des variables en jeu dans cette ligne afin de connaître la cause de l'erreur. Ensuite, il ne reste plus qu'à comprendre comment le programme en est venu à affecter cette valeur à cette variable. Cela peut se faire par une simple relecture du code, ou en utilisant le débogage pas à pas.

Prenons quelques exemples pour illustrer.

5.1 Pointeur non initialisé

Soit, le code d'exemple suivant :

```

1 #include <stdio.h>
2
3 typedef struct MaStruct
4 {
5     int foo;
6     int bar;
7 }MaStruct;
8
9 int main()
10 {
11     MaStruct* pStruct1;
12     MaStruct* pStruct2 = NULL;
13
14     pStruct1->foo = 10;
15     pStruct2->bar = 5;
16
17     return 0;
18 }
```

Les lignes `pStruct1->foo` et `pStruct2->bar` vont provoquer une erreur de segmentation. Plus précisément, le problème est que l'on essaie d'accéder à un membre d'une structure, mais cette structure n'a pas été définie en mémoire (pas d'allocation de mémoire associée).

Un débogueur s'arrêtera directement sur la ligne en question. Si on regarde le contenu des variables, `pStruct1` aura une valeur aléatoire et `pStruct2`, car elle est définie à la déclaration, vaudra `NULL`.



Sur certains systèmes (ou avec certains compilateurs), `pStruct1` peut, lui aussi, être initialisé à `NULL`. Mais, comme cela n'est pas un comportement généralisé, il vaut mieux déclarer tout pointeur à `NULL`.



Comme une variable qui n'est pas initialisée peut avoir une valeur aléatoire, il est **vivement** conseillé de lui donner une valeur à la déclaration. Cela vous évitera bien des erreurs. Notamment, il est facile de savoir l'état d'un pointeur (valide ou pas) si celui-ci est à `NULL` ou pas. Chose qui ne serait pas possible si vous ne définissiez pas vos pointeurs à `NULL` lors de la déclaration.



Sachez que si vous utilisez un compilateur correctement configuré (ayant les avertissements activés au niveau maximal : options « `-Wall -Wextra` » pour GCC), le compilateur vous retournera un message indiquant que vous utilisez `pStruct1` alors que celui-ci n'a pas été initialisé.

Si vous corrigez l'avertissement du compilateur, vous corrigez un bogue, c'est pour cela qu'il est fortement conseillé d'activer les **avertissements et d'en prendre compte**.

Un code corrigeant les erreurs pourrait être le suivant :

```

1 #include <stdio.h>
2
3 typedef struct MaStruct
4 {
5     int foo;
6     int bar;
7 }MaStruct;
8
9 int main()
10 {
11     MaStruct* pStruct1 = malloc(sizeof(
12         MaStruct));
13     MaStruct* pStruct2 = NULL;
14
15     if ( pStruct1 != NULL )
16     {
17         pStruct1->foo = 10;
18     }
19
20     if ( pStruct2 != NULL )
21     {
22         pStruct2->bar = 5;
23     }
24
25     free(pStruct1);
26
27     return 0;
28 }
```

Cette fois, j'ai alloué de la mémoire pour `pStruct1`. Pour `pStruct2`, je corrige l'erreur en vérifiant si mon pointeur est à `NULL` ou non (s'il est valide ou non).

Notez que je vérifie aussi la validité de `pStruct1`, car celui-ci pourrait valoir `NULL` si `malloc()` ne trouve pas assez de mémoire disponible pour allouer ma structure.

5.2 Pointeur non initialisé, le retour

Un autre cas de pointeur non initialisé est le suivant :

```

1 #include <stdio.h>
2
3 typedef struct MaStruct
4 {
5     int foo;
6     int bar;
7 }MaStruct;
8
```

```

9 void maFonction(MaStruct* pStruct)
10 {
11     pStruct->foo = 100;
12 }
13
14 int main()
15 {
16     MaStruct* pStruct1 = NULL;
17     maFonction(pStruct1);
18
19     return 0;
20 }
21

```

Le crash se produira dans la fonction `maFonction()`. Le problème est que la variable passée à votre fonction n'est pas nécessairement gérée par vous. Imaginons que vous travaillez à plusieurs et que par malheur un de vos collègues passe un pointeur `NULL` à votre fonction, le crash n'est pas totalement de votre faute.

Toutefois, il l'est un peu et vous pouvez faire en sorte de faire mieux. En effet, vous pouvez toujours vérifier si le pointeur passé est `NULL`. Pour cela, il existe les assertions implémentées dans la fonction `assert()` : [lien 90](#). Il faut savoir que la fonction `assert()` est une fonction qui ne sera exécutée que si le programme est compilé en « debug ». La fonction accepte comme argument un test. Si celui-ci correspond à zéro (ou faux) alors la fonction arrête le programme et affiche une ligne explicite de l'endroit où il s'est arrêté. Sinon, le programme continue tranquillement.

Voici un code corrigeant l'erreur, tout en utilisant l'assertion :

```

1 #include <stdio.h>
2
3 #include <assert.h>
4
5 typedef struct MaStruct
6 {
7     int foo;
8     int bar;
9 }MaStruct;
10
11 void maFonction(MaStruct* pStruct)
12 {
13     assert(pStruct);
14
15     pStruct->foo = 100;
16 }
17
18 int main()
19 {
20     MaStruct struct1;
21
22     maFonction(&struct1);
23
24     return 0;
25 }

```

Comme expliqué, j'ai rajouté un `assert()` afin de m'assurer que le pointeur passé à ma fonction n'est pas `NULL`.

La correction consiste à définir la structure statiquement (afin d'éviter de gérer la mémoire liée au pointeur) et de passer un pointeur sur cette structure, grâce au « & » à la fonction.



La plupart des bogues arrivent à cause des pointeurs. Il est donc préférable de les éviter, comme montré dans cet exemple.



L'assertion est certes un mécanisme fort, mais il ne faut pas l'utiliser pour, par exemple, vérifier des cas d'erreurs (retours de fonctions). En effet, `assert()` n'étant présent que si le programme est compilé en « debug », lors de la publication de votre programme, les assertions seront enlevées par le compilateur. Il ne remplace donc pas les tests que vous faites, mais vous protège juste des erreurs de programmation.

5.3 Double free

Un autre cas de crash survient lorsque l'on essaie de supprimer deux fois le même espace mémoire :

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int* pTab = malloc(sizeof(int) * 10)
7     ;
8
9     pTab[5] = 42;
10
11     free(pTab);
12     free(pTab);
13
14     return 0;
15 }

```

Le deuxième `free()` causera un crash.

Pour corriger le problème, je propose le code suivant :

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int* pTab = malloc(sizeof(int) * 10)
7     ;
8
9     pTab[5] = 42;
10
11     free(pTab); pTab = NULL;
12     free(pTab);
13
14     return 0;
15 }

```

Ce code fonctionne, car un `free()` d'un pointeur `NULL` ne fait rien. Ainsi, le conseil est donc de remettre le pointeur à la valeur `NULL` une fois que la mémoire est libérée. D'une part, cela permet de savoir si le pointeur est valide ou non et cela peut permettre d'éviter des erreurs.

6 Fuite de mémoire

Les fuites de mémoire font parties des problèmes spécifiques aux langages qui laissent une gestion partielle ou totale de la mémoire au programmeur. Le C et C++ sont les premiers programmes touchés par les fuites de mémoire.

En elle-même, la fuite de mémoire n'altère pas directement le comportement de votre programme. Toutefois, si votre programme consomme toujours plus de mémoire, sans se stabiliser, il est certain qu'un jour vous allez atteindre les limites de la machine et finir par crasher. Ainsi, les fuites sont une plaie pour tout programme qui ne doit jamais être arrêté et continuer pour des années (cas des programmes dans les systèmes embarqués, comme ceux pour les satellites).

Concrètement, qu'est-ce qu'une fuite de mémoire ? La fuite consiste en un pointeur (donc une zone mémoire) qui n'est plus accessible (la variable pointeur a été remplacée ou perdue).

Sous forme de code, ce qui suit a une fuite de mémoire :

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int* pInt = malloc(sizeof(int));
7
8     pInt = NULL; // Fuite
9     // Car nous n'avons plus la
10    // possibilité de faire un free
11    // de l'espace mémoire retourné par
12    // le malloc
13
14    return 0;
15 }
```

Ainsi, une fuite de mémoire apparaît à chaque fois qu'un malloc() n'a pas de free() correspondant.

Dans un grand programme, les fuites de mémoire peuvent devenir impossibles à trouver, il est donc nécessaire d'utiliser un programme pour nous aider. Je parle ici, de valgrind (lien 91) ou de Dr. Memory (lien 92).

6.1 Valgrind

Valgrind n'est malheureusement disponible que sous Linux. Son comportement est particulier, car il va exécuter le programme tout en vérifiant tous les accès mémoire. Dès qu'un de ces accès est invalide, il affiche une erreur. Lorsqu'il a fini, il va faire un résumé des allocations/désallocations du programme et indiquer le nombre d'octets perdus (les fuites).

L'utilisation de Valgrind est simple :

```

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

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

```

1 LEAK SUMMARY:
2 ==19814==    definitely lost: 92 bytes
3               in 4 blocks
4 ==19814==    indirectly lost: 25,664
5               bytes in 11 blocks
6 ==19814==    possibly lost: 8,176
7               bytes in 1 blocks
8 ==19814==    still reachable: 120,614
9               bytes in 1,348 blocks
10 ==19814==    suppressed: 0 bytes in
11            0 blocks
```

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

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

```

1 ==19814== 25,544 (56 direct, 25,488
2   indirect) bytes in 1 blocks are
3   definitely lost in loss record 169
4   of 170
5 ==19814==   at 0x4C27CC1: operator new(
6   unsigned long) (vg_replace_malloc.c:
7   261)
8 ==19814==   by 0x407C03: NE::SDL_Engine
9   ::initAPI() (SDL_Engine.cpp:65)
10 ==19814==   by 0x4042D7: NE::NEngine::
11   init() (NEngine.cpp:43)
12 ==19814==   by 0x444C92: main (main.cpp
13   :49)
```

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



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



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

6.2 Dr. Memory

Dr. Memory est disponible aussi bien sous Linux que sous Windows. Son utilisation et son fonctionnement sont très proches de ceux de Valgrind.

Pour le lancer, il suffit d'écrire dans une invite de commandes :

```
1 drmemory -show_reachable monProgramme
```

ou encore de glisser votre exécutable sur l'icône de Dr. Memory.

Une fois l'exécution terminée, Dr. Memory affiche son verdict. À la fin de celui-ci, vous trouverez un récapitulatif des fuites de mémoire :

```
1 ERRORS FOUND:
2      0 unique,      0 total
   unaddressable access(es)
3      0 unique,      0 total
   uninitialized access(es)
4      2 unique,      2 total invalid heap
   argument(s)
5      0 unique,      0 total GDI usage
   error(s)
6      0 unique,      0 total handle leak(
   s)
7      0 unique,      0 total warning(s)
8      1 unique,      1 total,      40 byte
   (s) of leak(s)
9      0 unique,      0 total,      0 byte
   (s) of possible leak(s)
10     0 unique,      0 total,      0 byte
   (s) of still-reachable
   allocation(s)
```

- unaddressable access : indique un accès (lecture ou écriture) à de la mémoire non allouée par votre programme ;

7 Explications supplémentaires

Quel que soit le langage que vous utilisez, il arrive toujours un moment où vous allez produire une erreur de segmentation (segmentation fault, ou segfault). Même si dans certains langages (Java, C#...) il est plus difficile de les produire, les erreurs restent possibles.

7.1 Les erreurs de segmentation en Java

Précédemment, nous avons vu les erreurs de segmentation. En Java, elles existent aussi, mais sont représentées par des exceptions. Reprenons nos cas d'erreurs de segmentation :

- une variable pointeur n'est pas initialisée et donc, pointe « n'importe où ». Ce cas n'est pas possible en Java ;

- uninitialized access : indique la lecture d'une donnée qui a été allouée, mais pour laquelle aucune valeur ne lui a été définie ;
- invalid heap argument : indique les pointeurs passés aux fonctions liées à la mémoire (free(), realloc(...)), mais qui ne pointent sur aucune zone mémoire valide ;
- GDI usage error : erreurs liées à l'utilisation de la bibliothèque GDI (Windows) ;
- handle leak : les identifiants de ressources (handles) qui sont toujours ouverts à la fin de l'exécution du programme ;
- leak : indique les fuites de mémoire ;
- possible leak : indique les fuites de mémoire liées aux pointeurs qui pointent sur le milieu d'un segment mémoire et non sur le début ;
- still-reachable : indique de la mémoire qui est toujours accessible.

Dans le rapport, chaque erreur est détaillée :

```
1 Error #3: LEAK 40 direct bytes 0x02050f9
   0-0x02050fb8 + 0 indirect bytes
2 # 0 replace_operator_new_array
   [d:\drmemory_package\
   common\alloc_replace.c:2642]
3 # 1 main
   [d:\developpement\sFML\sFML 2.2\
   sFML_2_2_template_vs2013\src\main.
   cpp:9]
```

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



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

- vous accédez au onzième élément, alors que votre tableau ne contient que dix éléments. Dans un tel cas, l'exception IndexOutOfBoundsException sera levée ;
- une variable pointeur est à NULL. Dans un tel cas, l'exception NullPointerException sera levée.

7.2 Mon bogue n'est pas reproductible dans le débogueur !

Il arrive qu'un bogue ne puisse pas être reproduit lorsque vous lancez votre programme dans le débogueur. Voici quelques pistes pour ce phénomène :

- vous n'avez pas initialisé une variable. La compilation avec les options de débogage fait que

le compilateur initialise les variables à zéro, même si vous ne le spécifiez pas. Le débogueur peut agir de même. Il est donc important de **toujours** initialiser les variables pour éviter de tomber dans ce cas ! ;

- vous avez une corruption de la mémoire (accès hors d'un tableau, modification de la mémoire involontaire...) Il arrive que les débogueurs soient un peu plus laxistes et laissent le programme continuer. Dans ce cas, utilisez un analyseur mémoire, pour vérifier que le programme ne fait pas n'importe quoi ;

8 Conseils supplémentaires

Tout le monde fait des bogues. C'est lié à la différence de réflexion et de compréhension du programme entre l'être humain et la machine. Toutefois, depuis le temps, les développeurs ont mis en place de nombreuses techniques pour repérer les bogues au plus tôt. Nous citerons :

- les avertissements du compilateur : le compilateur est capable de vérifier le code et de pointer quelques soucis dans son écriture. Alors, n'hésitez pas, activez tous les avertissements que le compilateur peut vous fournir ;
- les assertions : les assertions sont des mécanismes qui peuvent permettre de vérifier que l'état du programme est bien celui attendu. Attention, ces mécanismes ne sont activés que lorsque le programme est compilé en « debug » ;
- les analyseurs statiques : ils viennent en com-

- vous utilisez des threads et vous avez une race condition. L'exécution du programme étant plus lente dans le débogueur (même si cela n'est pas perceptible), la race condition ne se produit pas. Dans un tel cas, les débogueurs peuvent avoir des options supplémentaires pour mieux gérer les threads. Une relecture du code, une réflexion sur l'algorithme et l'utilisation de mécanisme de synchronisation entre threads permettront de supprimer le bogue.

plément du compilateur et se spécialisent dans la recherche des soucis en analysant le code ;

- les tests unitaires : ce sont des tests que vous allez programmer vous-même, permettant de vérifier le bon fonctionnement de vos fonctions. Ainsi, vous pourrez valider votre code au plus tôt ;
- les bonnes pratiques de programmation. De nombreuses pratiques existent pour éviter des bogues courants. C'est ce que l'on appellera : écrire du code propre. L'une de celles-ci est de toujours donner une valeur à vos variables. Vous pouvez en découvrir d'autres dans cet article : [lien 93](#).

Chaque langage propose ses propres outils adaptés. N'hésitez donc pas à vous renseigner sur le langage de programmation que vous utilisez.

9 Annexe : GDB

9.1 Démarrage de GDB

Ouvrez un terminal dans votre répertoire de travail pour votre application.

Tapez

```
1 gdb
```

Maintenant GDB est lancé. Celui-ci propose un environnement de travail dans lequel vous pouvez taper des commandes. Votre premier réflexe est de taper :

```
1 help
```

Auquel le programme répond :

```
1 List of classes of commands:
2
3 aliases -- Aliases of other commands
4 breakpoints -- Making program stop at
   certain points
5 data -- Examining data
6 files -- Specifying and examining files
7 internals -- Maintenance commands
```

```
8 obscure -- Obscure features
9 running -- Running the program
10 stack -- Examining the stack
11 status -- Status inquiries
12 support -- Support facilities
13 tracepoints -- Tracing of program
   execution without stopping the
   program
14 user-defined -- User-defined commands
15
16 Type "help" followed by a class name for
   a list of commands in that class.
17 Type "help all" for the list of all
   commands.
18 Type "help" followed by command name for
   full documentation.
19 Type "apropos word" to search for
   commands related to "word".
20 Command name abbreviations are allowed
   if unambiguous.
```

Vous savez accéder à l'aide. Généralement, il suffit de taper « help nom_de_la_commande » pour avoir une description de celle-ci sur la manière de

l'utiliser.

Pour arrêter sur GDB, il suffit de taper :

```
1 quit
```



Par la suite, j'indiquerai les commandes à taper dans le terminal avec le caractère « \$ » (qui n'est pas à recopier). Pour les commandes pour GDB, j'utiliserai « (gdb) ».

Pour un premier essai, ce n'est pas mal, mais ce qui nous intéresse, c'est de charger un programme à déboguer dans GDB. Il existe deux méthodes, une directement en lançant GDB :

```
1 $ gdb mon_programme
```

L'autre dans l'environnement de GDB :

```
1 (gdb) exec-file mon_programme
```

Si tout se passe bien, GDB affiche :

```
1 Reading symbols from mon_programme...
   done.
```

Mais il se peut qu'il affiche :

```
1 Reading symbols from mon_programme...(no
   debugging symbols found)...done.
```

Indiquant que vous n'avez pas compilé les programmes avec les informations de débogage. Dans ce cas, il suffit de recompiler le programme avec l'option permettant l'inclusion de ces informations. Si vous utilisez un EDI, bien souvent, il suffit de compiler avec la cible « debug ». Sinon, avec GCC il faut préciser l'option « -g ».

9.2 Lancement du programme dans GDB

Nous savons charger notre programme dans GDB, donc nous pouvons commencer à travailler dessus.

La première chose à faire, c'est de lancer le programme :

```
1 (gdb) run
```

Si vous souhaitez passer des paramètres supplémentaires à votre programme, vous pouvez très bien le faire juste après la commande run :

```
1 (gdb) run 42 deuxieme_parametre
```

Pour un programme normalement qui ne crashe pas, GDB affiche :

```
Starting program : mon_programme
Hello World
Program exited normally.
```

Le « Hello World » étant ce que mon programme a affiché de lui-même. À ce point, le programme est arrêté, donc il ne vous sera pas possible de vérifier

les valeurs des variables, ou de faire une exécution pas à pas.

Par contre, pour un programme qui produit une erreur de segmentation, GDB affichera des messages similaires à ceux-ci :

```
Program received signal SIGABRT, Aborted. 0x00007ffff7a8da75 in
*_GI_raise (sig=<value optimised out>) at ../nptl/sysdeps/unix/sysv/linux/raise.c :64 64.
../nptl/sysdeps/unix/sysv/linux/raise.c : Aucun fichier ou dossier de ce type. in
../nptl/sysdeps/unix/sysv/linux/raise.c
```

On y apprend que mon programme a reçu un signal « SIGABRT » venant du système. En effet, mon programme effectuant une opération invalide se fait immédiatement éjecter du système. C'est ce même signal qui fait que le programme se ferme sans prévenir, mais l'avantage du débogueur, c'est que le signal est reçu par GDB et fait en sorte de laisser votre application en vie. En plus, il indique d'où provient l'erreur et, comme nous allons le voir par la suite, nous pouvons remonter à la ligne provoquant toute cette zizanie. Pour l'instant, la seule information lisible, c'est que le signal vient de raise.c, mais avant de soupçonner un bogue dans le noyau du système ou dans une quelconque bibliothèque, il serait judicieux de vérifier notre programme.

Pour information, le programme qui a effectué ce crash est le suivant :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int* tab = malloc(sizeof(int)*10);
7
8     unsigned int i = 0;
9     for ( i = 0 ; i < 11 ; i++ )
10    {
11        tab[i] = i;
12    }
13
14    free(tab);
15
16    return 0;
17 }
```



L'interprétation des commandes par GDB est évoluée. Par exemple, si vous lancez une fois votre programme avec la commande « run arg1 arg2 », un appel à la commande « run » sera interprété comme rappel de la commande « run arg1 arg2 ». De même que pour les commandes longues, telles que « backtrace », il existe le raccourci « bt ».

Retrouvez l'article d'**Alexandre Laurent** en ligne : [lien 94](#)

2D/3D/Jeux

Les derniers tutoriels et articles

Vulkan, la nouvelle bibliothèque de hautes performances pour le GPU

Découvrez la nouvelle bibliothèque pour le GPU de Khronos. Celle-ci se place comme successeur d'OpenGL.

1 Introduction

Vulkan est le nom de la nouvelle bibliothèque de hautes performances pour le GPU. D'abord, intéressons-nous à l'histoire pour mieux comprendre les raisons qui nous ont apporté Vulkan.

1.1 Historique

OpenGL, pour **Open Graphics Library**, a été conçu par Silicon Graphics Inc. en 1991 et la première version publiée date de janvier 1992. OpenGL est une bibliothèque visant à exposer de manière unifiée les fonctionnalités des cartes graphiques. Un de ses grands avantages est d'être indépendante du langage et du système d'exploitation.

En 1995, Microsoft publie Direct3D qui devient la principale concurrente d'OpenGL.

En 2004 arrive la version 2.0 d'OpenGL, apportant le support des shaders et implémentant le **GLSL** (OpenGL Shading Language). Les shaders ont révolutionné le rendu qui, jusqu'à présent, était préprogrammé statiquement dans les cartes graphiques. Avec les shaders, le programmeur a la possibilité de programmer lui-même une partie du rendu effectué par la carte graphique et ainsi créer ses propres effets et optimiser le résultat.

En plus de l'intégration des shaders, OpenGL a mis à disposition les tampons de sommets (*vertex buffer*) et des alternatives au mode immédiat (le fait de donner des ordres un par un pour décrire les géométries, notamment avec les blocs `glBegin()` (lien 95)/`glEnd()` (lien 96)).

En 2006, le contrôle du standard est transféré à Khronos (lien 97), un consortium regroupant les acteurs du monde 3D (NVIDIA, AMD, Intel, Apple, Imagination Technologies...). Depuis, OpenGL n'a cessé de progresser et de rattraper un retard par rapport aux possibilités des cartes graphiques.

Avec OpenGL 3.0 publiée en 2008, un mécanisme de dépréciation a été mis en place. L'intégralité du

pipeline fixe, le mode immédiat, les listes d'affichage (*display list*) et le support des couleurs indexées ont été retirés. Malgré cette dépréciation, les fonctions sont toujours présentes et implémentées dans les pilotes.

En 2010, OpenGL 3.3 et OpenGL 4.0 sont publiées : lien 97. La première vise le même niveau de fonctionnalités que Direct3D 10, alors que la seconde vise les fonctionnalités de Direct3D 11. La version 4 d'OpenGL apporte la tessellation. Depuis, chaque année une nouvelle version mineure d'OpenGL voit le jour, apportant les dernières nouveautés disponibles sur les cartes graphiques.



Vous pouvez suivre toutes ces évolutions en allant sur le portail de la rubrique 2D/3D/Jeux : lien 98. Pour plus de facilité, n'hésitez pas à vous abonner au flux RSS (lien 99), Twitter (lien 100), Facebook (lien 101) de la rubrique.

1.2 Les problèmes d'OpenGL

OpenGL a plus de vingt ans et depuis 1992, les cartes graphiques ont énormément évolué. Une des évolutions les plus marquantes est l'apparition des shaders. Bien qu'OpenGL 4.X intègre toutes les dernières nouveautés, le support d'OpenGL 1.0 est toujours inclus dans les pilotes graphiques. De plus, l'architecture intrinsèque d'OpenGL n'a pas changé depuis 1992 (machine à états) causant des soucis d'évolutions. En 2014, un développeur avait listé les problèmes de la bibliothèque : lien 102. On peut citer :

- les vingt ans d'héritage ont rendu la bibliothèque complexe ;
- la bibliothèque ne gère pas le multi-threading ;

- la compilation des shaders dépend de l'implémentation du constructeur ;
- l'absence d'outils standards/officiels pour OpenGL ;
- ...

1.3 La naissance d'une nouvelle ère

Fin 2013, AMD a dévoilé une nouvelle bibliothèque graphique : Mantle ([lien 103](#)). Celle-ci possède la particularité de donner aux développeurs un meilleur contrôle sur le GPU. En effet, avec les bibliothèques comme OpenGL ou encore Direct3D 11, le pilote n'a aucun moyen de savoir exactement ce que veut le développeur, et donc il n'a aucun moyen d'optimiser le processus. En donnant un accès plus bas niveau au GPU, les développeurs peuvent réaliser exactement ce qu'ils veulent. De plus, le pilote accorde une plus grande confiance aux développeurs, allégeant ainsi la complexité de l'implémentation et son utilisation CPU. Finalement, Mantle

supporte le multi-threading. Peu de temps après, Microsoft présente Direct3D 12 ([lien 104](#)), reprenant les mêmes concepts, puis c'est au tour d'Apple avec Metal ([lien 105](#)). Finalement, Khronos a aussi annoncé une évolution de la bibliothèque OpenGL, nommée OpenGL Next ([lien 106](#)). Pour celle-ci, AMD a annoncé ([lien 107](#)) apporter tout son support et son expérience acquise avec Mantle.

1.4 Vulkan



Finalement, Khronos a dévoilé Vulkan ([lien 108](#)), le successeur d'OpenGL, au cours de la Game Developers Conference 2015. Voyons ce qu'il en est vraiment.

2 Fonctionnalités

L'objectif de Vulkan est de créer une toute nouvelle bibliothèque (sans avoir à supporter d'anciennes fonctionnalités) pour les cartes récentes et ayant l'objectif de combler les lacunes d'OpenGL.

Ainsi, Vulkan apporte :

- **un contrôle direct du GPU avec un impact minimal sur les performances par le pilote.** Un exemple de cela est l'écriture des données directement sur le GPU au lieu de passer par des appels tels que `glUniform()` : [lien 109](#). Ainsi les applications peuvent implémenter leur propre stratégie d'allocation ;
- **les passes de rendu** (*render pass*) permettant de contrôler le chargement des cibles de rendu au début et à la fin du rendu ;
- **une architecture compatible avec le multi-thread.** Les tampons de commandes peuvent être remplis par plusieurs threads à

la fois et même être envoyés au GPU par un thread séparé ;

- **une bibliothèque unifiée pour les PC, les mobiles et les plateformes embarquées.** Il n'est plus question d'avoir une version pour les PC (OpenGL) et une autre pour les mobiles (OpenGL ES). Vulkan est une seule bibliothèque pour toutes les plateformes ;
- **un code intermédiaire pour les shaders.** Ces derniers peuvent être maintenant envoyés au pilote au format bytecode SPIR-V : [lien 110](#). Khronos fournit un compilateur GLSL vers SPIR-V. Il sera possible à tout un chacun de faire son propre compilateur.

La finalité de Vulkan est de simplifier les pilotes, d'avoir des performances plus stables, un meilleur contrôle sur le GPU et moins de différences entre les implémentations des constructeurs.

3 Implémentation

Toutes cartes graphiques supportant OpenGL 4.3/OpenGL ES 3.1 ou supérieur pourront supporter Vulkan.

Voici un exemple de code avec Vulkan :

```
1 vkCmdBindDescriptorSet(cmdBuffer,
  VK_PIPELINE_BIND_POINT_GRAPHICS,
  textureDescriptorSet[0], 0);
```

```
2 vkQueueSubmit(graphicsQueue, 1, &
  cmdBuffer, 0, 0, fence);
3 vkMapMemory(staticUniformBufferMemory, 0
  , (void **)&data);
4 // ...
5 vkUnmapMemory(staticUniformBufferMemory
  );
```

4 Outils

Plusieurs sociétés (Valve, LunarG, Codeplay) travaillent déjà sur des outils. Voici GLAVE, un outil. *Developpez Magazine* est une publication de *Developpez.com*

til de débogage développé par Valve et LunarG : [lien 111](#)

5 Démonstrations

5.1 Imagination Technologies

Imagination Technologies a porté une de leurs démonstrations OpenGL ES 3.0 vers Vulkan : [lien 112](#)

La démonstration possède quelques effets graphiques en moins comparée à la version OpenGL ES 3.0 à cause de la contrainte de temps. Toutefois, dans celle-ci, vous pouvez voir :

- un rendu graphique basé sur la physique (*physically-based shading*) ;
- un rendu *High Dynamic Range* (HDR) ;
- vingt textures 2048 x 2048 au format propriétaire PVRTC ;
- deux gigaoctets de données compressées dans 266 mégaoctets grâce au format PVRTC ;
- 4 x *Mutil-sample anti-aliasing* (MSAA) ;
- un filtrage anisotropique 16x ;
- une utilisation basse du CPU et une utilisation efficace du GPU ;
- plus de 250 000 triangles ;
- des effets de post-traitements : saturation, exposition et *tone mapping*.

6 Conférence GDC 2015

Vulkan a été présentée durant la conférence Game Developers Conference 2015. Au cours de celle-ci, vous pourrez découvrir comment s'architecture la bibliothèque et comment l'utiliser. Vous y

7 Disponibilité

La spécification finale devrait être publiée au cours de l'année 2015. La plupart des constructeurs

8 Documentation

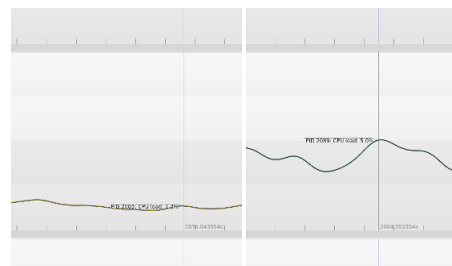
Site officiel : [lien 115](#).
 Présentation de Khronos à la GDC 2015 : [lien 116](#).
 Présentation de Valve à la GDC 2015 : [lien 117](#).

Retrouvez l'article d'**Alexandre Laurent** en ligne : [lien 120](#)

5.1.a Un CPU moins sollicité

Avec Vulkan, le CPU est moins sollicité. En effet, lorsque vous faisiez un `glUniform*()` avec OpenGL, le pilote graphique devait allouer de la mémoire sur le GPU pour accueillir les données. Avec Vulkan, vous récupérez simplement un pointeur sur cette mémoire et vous y copiez les données. Une seule et unique allocation est nécessaire, car vous savez exactement l'espace requis pour vos données et une seule copie est effectuée.

Voici la différence en termes d'utilisation CPU entre OpenGL ES et Vulkan :



5.2 Valve

Valve, grâce au pilote Linux open source d'Intel, a pu montrer une démonstration du moteur Source 2 utilisant Vulkan : [lien 113](#)

trouverez aussi une description détaillée de SPIR-V.

Voir le compte-rendu de la conférence sur Vulkan : [lien 114](#).

de pilotes ont d'ores et déjà annoncé que leurs pilotes seront prêts à ce moment-là.

Spécification du langage intermédiaire SPIR-V : [lien 118](#).

Introduction au langage intermédiaire SPIR-V : [lien 119](#).

Qt



Les dernières news Qt

Sortie de Qt 5.5 Beta

Qt 5.5 alpha est disponible depuis peu. Le principal objectif de cette nouvelle version est l'amélioration et la stabilisation des fonctionnalités existantes, mais de nouvelles fonctionnalités font aussi leur apparition. Avec Qt 5.5, Canvas 3D est totalement supporté et le très attendu Qt 3D est inclus en qualité d'avant-première technologique. Qt 5.5 facilite la cartographie au travers du nouveau module Qt Location, lui aussi en qualité d'avant-première technologique. Cette version alpha de Qt 5.5 est un premier pas vers la version finale de Qt 5.5, prévue pour mai, avec un mois de retard sur le planning initial.

Stabilisation et amélioration de l'existant

Les précédentes versions de Qt 5 ont apporté la compatibilité avec de nouvelles plates-formes ainsi que de nouvelles fonctionnalités. Avec Qt 5.5, les équipes de développement se sont concentrées sur la stabilité ainsi que l'amélioration des fonctionnalités existantes. La plupart des nouvelles fonctionnalités sont mineures ou portent des API existantes sur de nouvelles plates-formes. Qt 5 constitue une fondation solide et cela sera d'autant plus vrai avec Qt 5.5.

En plus des améliorations apportées à Qt en lui-même, les systèmes d'assurance qualité et d'intégration continue testent maintenant de plus en plus de plates-formes et de configurations différentes. Avec Qt 5.5, le système d'intégration continue et de publication permet maintenant de sortir de nouvelles versions de patch (Qt 5.5.x), même après la sortie de Qt 5.6.

Qt 3D et Qt Canvas 3D

Une des fonctionnalités les plus attendues de Qt 5.5 est la préversion de Qt 3D 2.0. De gros efforts ont été consentis à ce sujet, notamment de la part des développeurs de KDAB. Ce module Qt 3D 2.0 est maintenant prêt pour les tests. Depuis de nombreuses années, Qt a été utilisé pour développer des applications 3D de renommée, mais Qt 3D facilite plus que jamais l'intégration et l'utilisation de contenu 3D dans vos applications Qt. Ce module fournit une API C++ ainsi qu'une API QML pour intégrer du contenu 3D dans tout type d'application et permet d'utiliser OpenGL 2, 3 et 4 ainsi qu'OpenGL ES 2 et ES 3. Pour plus de détails sur ce qu'apporte Qt 3D 2.0, vous pouvez consulter la do-

umentation de Qt 3D (lien 121) ainsi que cette série d'articles publiés sur le blog de KDAB (lien 122).

En plus de Qt 3D, toujours dans le domaine de la 3D, Qt Canvas 3D (lien 123) sort, avec cette nouvelle version, de son statut d'avant-première technologique. Avec ce nouveau module léger, il devient très facile de faire des appels à des fonctions 3D à la « WebGL » depuis un contexte Qt Quick / JavaScript, permettant d'utiliser facilement des ressources WebGL par-dessus Qt Quick. Pour beaucoup de besoins dans le domaine 3D, cela permet de tirer parti d'applications écrites en HTML5 / WebGL.

Qt Location

Encore un autre module attendu depuis longtemps, ajouté à Qt 5.5 en qualité d'avant-première technologique, Qt Location (lien 124). Ce module apporte des fonctionnalités de cartographie, de géocodage, de géocodage inverse, de routage ainsi que de placement à Qt. Il est maintenant aisé d'utiliser des fonctions de cartographie dans des applications développées avec Qt. Ce module exploite les données géographiques de différents fournisseurs, comme Open Street Map, Mapbox ou Here Maps.

Qt Multimedia

Qt Multimedia utilise maintenant GStreamer 1.0 et ajoute de nombreuses nouvelles fonctionnalités. Une des fonctionnalités les plus intéressantes est un nouveau cadre de travail concernant le filtrage vidéo. Ce dernier facilitera l'intégration de bibliothèques comme OpenCV ou de bibliothèques de calcul comme OpenCL ou CUDA, via les éléments VideoOutput. Qt Multimedia se voit par ailleurs adjoindre une nouvelle API pour contrôler les paramètres du viseur et gère mieux la caméra sous iOS.

Qt Quick et les contrôles Qt Quick

Qt 5.5 introduit un nouveau contrôle, TreeView. Par ailleurs, tous les contrôles précédemment fournis dans la version commerciale (CircularGauge, DelayButton, Dial, Gauge, PieMenu, StatusIndicator, ToggleButton, Tumbler et TumblerColumn) sont maintenant disponibles dans la version libre. Ces anciens contrôles sont maintenant disponibles grâce à l'import QtQuick.Extras.

Le rendu dans un fil d'exécution distinct via QQuickRenderControl est maintenant possible. Sous Windows, Qt Quick est maintenant traité par défaut

dans la boucle de rendu parallèle de Qt Quick lorsqu'OpenGL est utilisé. Le pavé tactile d'OS X est maintenant mieux pris en charge, le pincement est maintenant géré via PinchArea, tout comme le zoom intelligent qui fait son apparition dans cette version.

Le moteur de Qt Quick gère maintenant les tableaux typés de JavaScript. Il est maintenant facile d'exposer des types C++ définis par l'utilisateur dans un contexte JavaScript / QML et dans le QJSEngine. En outre, le moteur Qt Quick a reçu différentes améliorations des performances.

Mise à jour de Qt WebEngine et de Qt WebView

Qt WebEngine a été mis à jour et contient maintenant la version 40 de Chromium, qui ajoute de nouvelles API. Qt WebEngine expose maintenant des API concernant le téléchargement de fichiers, la géolocalisation, le cache et les cookies, mais aussi les paramètres. Par ailleurs, de nombreuses API précédemment marquées comme expérimentales sont maintenant publiques. Qt WebChannel a été intégré dans la communication interprocessus de Chromium, facilitant et améliorant la sécurité des applications hybrides. Qt WebEngine met par ailleurs à disposition une nouvelle API pour les scripts utilisateurs, qui en combinaison avec Qt WebChannel facilite le développement de puissantes applications hybrides.

Qt WebView est maintenant implémentée nativement sous OS X (en plus d'Android et d'iOS). Qt WebView expose maintenant une API concernant les notifications de statuts, le contenu HTML ou bien le JavaScript en cours d'exécution.

Autres nouvelles fonctionnalités

Qt Bluetooth gère totalement le Bluetooth basse consommation, tant sur Android qu'iOS. Par ailleurs, le Bluetooth classique est maintenant supporté sous iOS. Le module Qt Network n'est pas en reste, avec une nouvelle implémentation de SSL pour iOS et OS X reposant sur le « Secure Trans-

port », les suites d'algorithmes TLS PSK, ainsi que les certificats à courbe elliptique.

Sous Linux, Qt NFC dispose maintenant d'une implémentation reposant sur *near* et différentes améliorations ont été apportées à QPA. Pour plus de détails, vous pouvez consulter la liste des nouvelles fonctionnalités de Qt 5.5 : [lien 125](#).

Configurations compatibles

Afin de mieux supporter Qt WebEngine ainsi que de nouvelles fonctionnalités profitant des nouveaux compilateurs, le système d'intégration continue et de distribution a été mis à jour pour Qt 5.5. Pour ajouter ces nouvelles configurations, d'autres plus anciennes devront être abandonnées, comme Ubuntu 11.10 et 12.04; OS X 10.7 ne sera supporté qu'en citoyen de seconde zone. Par contre, Qt 5.5 fonctionnera sur Windows 10 (lorsqu'il sera disponible) ainsi que RedHat Enterprise Linux 6.6.

Vous trouverez plus de détails sur la nouvelle configuration du système d'intégration continue sur le wiki : [lien 126](#).

Modules dépréciés

Les modules Qt WebKit, Qt Declarative (Qt Quick 1) et Qt Script sont maintenant dépréciés : en effet, Qt WebEngine est le remplaçant direct de Qt WebKit, Qt Quick 2 remplace Qt Quick 1 et les fonctionnalités de Qt QML remplacent celles de Qt Script. Pas de panique, tous ces modules sont toujours inclus dans Qt 5.5, mais seront supprimés dans une version ultérieure de Qt.

Obtenir Qt 5.5 Alpha

Les sources de Qt 5.5 alpha sont disponibles sur la page des téléchargements : [lien 127](#). Les fichiers binaires seront fournis lors de la sortie de la bêta, mais des instantanés seront quand même disponibles avant.

Aidez à parfaire Qt 5.5. Testez Qt 5.5 Alpha et signalez tout défaut sur l'application de suivi de bogues : [lien 128](#).

Commentez la news d'Arnold Dumas en ligne : [lien 129](#)

Sortie de Qt Creator 3.4.0

La nouvelle version de Qt Creator, numérotée 3.4.0, vient d'arriver, malgré les retards de Qt 5.5. Elle se focalise sur le peaufinage de l'existant, avec des corrections de défauts (notamment au niveau du débogueur) et des améliorations du code interne, tout en apportant quelques nouvelles fonctionnalités.

Côté C++, une nouvelle action de refactorisation a été ajoutée pour déplacer les définitions de fonction en dehors d'une définition de classe ; égale-

ment, l'autocomplétion propose maintenant la nouvelle syntaxe pour la connexion entre signaux et slots arrivée avec Qt 5. Un nouveau filtre propose également de signaler tous les fichiers C et C++ inclus dans le projet, même sans être explicitement mentionnés.

L'intégration Android est désormais compatible avec les chaînes de compilation 64 bits. Le développement sur des plateformes embarquées sans Qt (*bare metal*) peut être fait avec des projets géné-

riques.

Clang se fait une place plus importante dans l'EDI : son analyseur statique n'est plus considéré comme expérimental, il peut d'ailleurs être utilisé en combinaison avec les compilateurs Visual C++

et MinGW.

Ces nouveautés seront-elles suffisantes en regard de la nouvelle concurrence qui s'annonce dans le domaine des EDI C++ (comme CLion : [lien 130](#)) ?

Commentez la news de **Thibaut Cuvelier** en ligne : [lien 131](#)

Les derniers tutoriels et articles

Aperçu de Qt 3D 2.0

Présentation générale

Quand Qt était toujours un produit de Nokia, une équipe de développement de Brisbane, en Australie, a eu l'idée de faciliter l'incorporation de contenu 3D dans les applications Qt. Ces événements ont eu lieu en même temps que le développement du langage QML, il était donc naturel que Qt 3D ait une API QML, en plus d'une interface plus traditionnelle en C++, comme les autres modules de Qt.

1 Historique

La première version de Qt 3D a été publiée en même temps que Qt 4 et n'a été que peu utilisée avant que Qt ne passe aux mains de Digia. Durant l'opération, les bureaux de Brisbane ont été fermés : par conséquent, Qt 3D n'a jamais été adapté pour Qt 5. Sans mainteneur, le code est tombé à l'abandon.

OpenGL devenant de plus en plus important dans Qt 5 (notamment pour Qt Quick 2), mais également dans les projets réalisés avec Qt, il paraissait intéressant de reprendre en main le projet Qt 3D, de le ressusciter, mais également de le rendre plus compétitif par rapport aux solutions modernes de

3D.

Cet article est le premier d'une série qui couvrira les possibilités qu'offre ce renouveau, les API et leurs implémentations en détail. Les prochains articles montreront la manière d'utiliser les API pour différents buts, des plus simples aux plus compliqués, à l'aide d'exemples détaillés. Ce premier article donnera un aperçu général des objectifs poursuivis lors de la conception de Qt 3D, mais également des difficultés rencontrées et leur solution, ce qu'il reste à faire avant une finalisation de Qt 3D 2.0 et ce que le futur pourra amener comme progrès.

2 Objectifs de Qt 3D

La plupart des gens qui ne sont pas familiers avec le rendu 3D pensent qu'une brique logicielle telle que Qt 3D devrait être capable de dessiner des formes 3D et de les déplacer, ainsi que de gérer la caméra. C'est un bon début, mais, en creusant un peu plus loin, ils aimeraient également d'autres choses :

- la 2D autant que la 3D ;
- les maillages ;
- les matériaux ;
- les ombres.

Ensuite, en posant la même question au groupe suivant, ceux qui ont une certaine habitude des complexités de la 3D, les réponses sont plus techniques :

- les shaders : [lien 132](#) ;
- l'occlusion ambiante : [lien 133](#) ;
- le rendu à grande plage dynamique : [lien 134](#) ;
- le rendu ([lien 135](#)) différé ([lien 136](#)) ; différé
- l'utilisation de plusieurs textures pour un même polygone : [lien 137](#) ;
- l'instanciation : [lien 138](#) ;
- les UBO : [lien 139](#) ;
- toute nouvelle technique présentée au SIGGRAPH.

Cette liste est déjà fort complexe, mais la pire d'entre elles est la dernière : *l'utilisateur veut confi-*

gurer le rendu de manière actuellement inimaginable. Qt 3D, premier du nom, offrait des interfaces multiples, que ce soit pour le QML ou le C++ - et cette fonctionnalité reste intéressante à garder. Le concept de « graphe de trame » respecte toutes ces contraintes, c'est-à-dire qu'il permet de configurer complètement le rendu.

Là où le graphe de scène était une description orientée données du contenu à afficher, le graphe de trame est une description orientée données de la manière d'afficher.

Utiliser une description orientée données donne une grande liberté de choix : un simple rendu non différé, avec une profondeur, un affichage différé ou d'objets transparents, etc. Puisque toute la chaîne est configurée à l'aide de données, non de code, tout peut changer très facilement, même de manière dynamique durant l'exécution, sans toucher à du code C++.

Après cet amuse-bouche — afficher du contenu 3D à l'écran —, les utilisateurs voudront aussi effectuer des opérations intéressantes sur ces objets 3D. La liste est longue et variée, mais les points qui reviennent le plus souvent sont :

- la simulation physique : [lien 140](#) ;
- la détection de collisions : [lien 141](#) ;
- l'audio 3D : [lien 142](#) ;
- l'animation de corps rigides, de squelettes ([lien 143](#)), par forme cible ([lien 144](#)) ;

3 Vue d'ensemble de l'architecture

Tous ces besoins se sont révélés être épineux, très épineux à concilier. Heureusement, il a été possible de trouver des solutions abordables pour la plupart des problématiques.

Les commentaires sur leur résolution commenceront à un très haut niveau : comment implémenter un module Qt suffisamment extensible pour gérer non seulement l'affichage, mais également les autres fonctionnalités — et d'autres encore, impossibles à prévoir ?

Le cœur de Qt 3D est de simuler des objets en temps réel puis, très probablement, d'afficher l'état de ces objets à l'écran, d'une manière ou d'une autre. Cet énoncé propose déjà une question intéressante : qu'est-ce qu'un objet ?

3.1 Notion d'objet

Bien évidemment, dans ce genre de système de simulation, le nombre de types différents d'objets sera probablement très grand. Un jeu simple, comme Space Invaders ([lien 147](#)), peut déjà en comporter un grand nombre. Une application réelle sera très certainement beaucoup plus complexe, mais ce jeu

- la recherche de chemin ([lien 145](#)) et d'autres techniques d'intelligence artificielle ([lien 146](#)) ;
- la détermination de l'objet sélectionné par l'utilisateur ;
- les particules ;
- etc.

Il serait bien évidemment impossible de répondre à toutes ces demandes avec des ressources limitées pour le développement. Cependant, il est clair que, pour incorporer toutes ces fonctionnalités dans le futur, l'architecture de Qt 3D 2.0 doit être extensible et flexible. Les travaux préparatoires ont été longs et beaucoup de prototypes ont été développés avant d'atteindre l'architecture actuelle. Elle sera présentée plus loin dans cet article et développée dans un prochain.

En sus de ces objectifs à plus ou moins long terme, Qt 3D devait également avoir de bonnes performances et s'adapter au nombre de cœurs disponibles. Ce dernier point est important à cause de la manière actuelle d'améliorer les performances du matériel : en augmentant le nombre de cœurs plutôt que la fréquence. D'ailleurs, en analysant ces demandes, intuitivement, il semble possible d'exploiter plusieurs cœurs, puisque bien des tâches sont indépendantes les unes des autres. Par exemple, les opérations de recherche de chemin ne chevauchent pas les tâches de rendu (sauf peut-être pour l'affichage d'informations de débogage ou de statistiques).

pointera déjà une série de questions. Parmi les objets très probablement présents dans un tel jeu, on peut compter :

- le canon du joueur, au sol ;
- le sol ;
- les blocs de défense ;
- les vaisseaux de l'ennemi ;
- la soucoupe volante du boss ;
- les balles, tirées tant par le joueur que par l'ennemi.



Dans une conception de la plus pure tradition C++, ces objets seront très probablement implémentés en utilisant des classes, dans une certaine

hiérarchie d'héritage. Les différentes branches pourront apporter des fonctionnalités aux objets, comme « accepte une entrée de la part de l'utilisateur », « émet des sons », « peut être animé », « entre en collision avec d'autres objets », « doit être affiché à l'écran ». Cependant, concevoir une telle hiérarchie, même pour un problème aussi simple, n'est pas facile.

Cette approche et d'autres variations sur le thème de l'héritage ont un certain nombre de problèmes (qui seront discutés plus en détail dans d'autres articles) :

- une hiérarchie profonde et large est difficile à comprendre, à maintenir et à étendre ;
- la taxonomie d'héritage est définie à la compilation et ne peut plus être changée ;
- chaque niveau de la hiérarchie ne peut classifier que selon un critère ;
- les fonctionnalités partagées tendent à monter dans la hiérarchie avec le temps ;
- le concepteur de la bibliothèque ne peut jamais savoir tout ce que les utilisateurs feront.

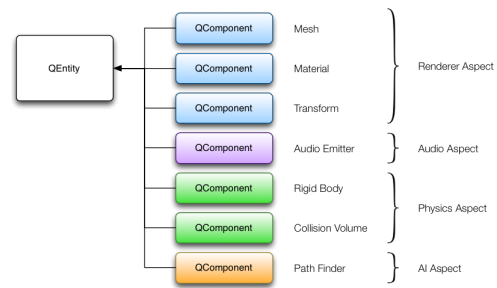
Toute personne qui a travaillé avec une telle hiérarchie trouvera probablement que, à moins de comprendre et d'être d'accord avec la taxonomie de l'auteur, il peut être très difficile de l'étendre sans bidouiller pour y faire rentrer les extensions requises.

3.2 Système à entités

Pour Qt 3D, le concept d'héritage a été en grande partie abandonné et remplacé par l'agrégation en tant que manière d'importer des fonctionnalités dans une instance d'un objet. Plus précisément, Qt 3D utilise un système à entités (ECS : *entity-component system*). Ce principe peut être implémenté de diverses manières et les détails spécifiques à Qt 3D seront abordés dans un prochain article.

Une *entité* représente un objet simulé, mais est, par elle-même, dénuée de tout comportement ou caractéristique. Un comportement peut être greffé sur une entité en agrégeant des *composants*.

Ainsi, un composant est une partie de comportement ou de fonctionnalité, dans la même veine que ceux décrits pour le Space Invaders. Par exemple, le sol serait une entité attachée à un composant qui indique au système qu'il faut l'afficher, mais également comment l'afficher. Un ennemi sera une entité attachée à un composant qui indique qu'il faut l'afficher à l'écran, un autre pour l'émission de sons, encore un autre pour les collisions, l'animation, un dernier pour l'intelligence artificielle. Le joueur sera fort similaire à l'ennemi, à l'exception qu'il n'aurait pas d'intelligence artificielle, plutôt un composant gérant les entrées du joueur (les déplacements et les tirs).



Qt 3D implémente la partie système du paradigme, sous la forme d'aspects. Un aspect implémente une partie des fonctionnalités attribuées aux entités par une agrégation d'un ou plusieurs composants. Concrètement, l'aspect du rendu recherche les entités qui ont un maillage, une texture et des composants de transformation ; s'il trouve une telle entité, il sait comment exploiter ces données et effectuer un rendu : par conséquent, il ignore les entités qui n'ont pas ces composants.

De même, un aspect gérant la physique pourrait chercher des entités qui ont une certaine forme de volume de collision comme composant, ainsi qu'un autre composant qui spécifie d'autres propriétés requises pour la simulation, comme la masse ou le coefficient de friction. Une entité qui émet un son pourrait avoir un composant qui indique quel son jouer et à quel moment.

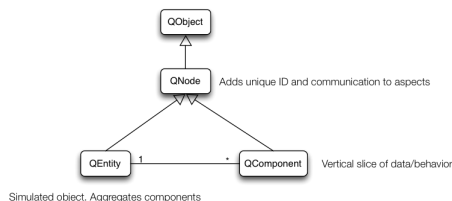
En résumé, Qt 3D construit des *entités* en agrégeant des *composants* qui leur communiquent des aptitudes. Le moteur Qt 3D utilise des aspects pour traiter et mettre à jour les entités avec des composants spécifiques.

Une fonctionnalité très intéressante d'un système à entités est que, par l'utilisation de l'agrégation en lieu et place de l'héritage, il est possible de changer, dynamiquement, le comportement d'un objet, simplement en ajoutant ou en retirant des composants. Par exemple, pour laisser le joueur parfois traverser les murs, il suffit de retirer le composant gérant le volume de collision de l'entité, puis de le remettre quand la durée du pouvoir spécial est écoulée. Ce système n'impose donc pas de créer une classe comme `JoueurQuiPeutParfoisTraverserLesMurs`.

3.3 Implémentation

Ceci donne une bonne indication du niveau de flexibilité du système à entités, c'est d'ailleurs pour cela qu'il a été choisi comme fondation architecturale de Qt 3D. La hiérarchie de classes correspondante est également très simple. La « classe de base » de Qt 3D est `QNode`, un dérivé très simple de `QObject` qui gère automatiquement la communication des changements de propriétés à travers le système d'aspects et un identifiant unique dans l'application. Un futur article expliquera les détails des aspects : ils travaillent dans des fils d'exécution séparés, `QNode`

simplifie fortement le transfert de données entre les objets visibles à l'utilisateur et les aspects. Souvent, les sous-classes de QNode fournissent des données supplémentaires, qui sont ensuite référencées par les composants. Par exemple, un QShaderProgram spécifie du code GLSL à utiliser pour l'affichage d'un ensemble d'entités.



Les composants de Qt 3D sont implémentés en héritant de QComponent et en ajoutant toute information nécessaire pour que les aspects correspondants effectuent leur travail. Par exemple, le composant de maillage Mesh est utilisé par l'aspect de rendu pour récupérer les informations de chaque sommet à envoyer à OpenGL.

Finalement, QEntity est un objet qui agrège simplement un ensemble d'au moins zéro QComponent, comme expliqué plus haut.

4 Résumé

Les cas d'utilisation de Qt 3D dépassent largement l'implémentation d'un moteur de rendu non configurable, mais accessible depuis QML. Il s'agit plutôt d'une solution de rendu complètement configurable, qui permet d'implémenter rapidement toute technique de rendu souhaitée. De plus, Qt 3D fournit un environnement générique pour des simulations en temps (presque) réel au-delà du rendu. Qt 3D sépare proprement le cœur d'un nombre indéfini d'aspects qui peuvent implémenter n'importe quel type

Pour ajouter une nouvelle fonctionnalité à Qt 3D, en tant que partie de Qt ou extension spécifique à une application, tout en bénéficiant des aptitudes de parallélisation du système, il suffit de suivre quelques étapes :

- identifier et implémenter les composants nécessaires pour gérer les données ;
- enregistrer ces composants auprès du moteur Qt Quick (uniquement pour exploiter l'API QML) ;
- hériter de QAbstractAspect et implémenter les fonctionnalités du nouveau sous-système.

La gestion du parallélisme est native pour Qt 3D, vu comme un moteur orienté tâches : il demande à chaque aspect, pour chaque trame à dessiner, une liste de tâches à exécuter, ainsi que leurs dépendances. Les tâches sont alors distribuées sur les différents cœurs de calcul de la machine par un ordonnanceur.

Dit comme cela, la chose peut paraître aisée. Après avoir implémenté l'aspect de rendu et effectué quelques recherches sur d'autres aspects potentiels, l'équipe de développement est assez sûre d'elle : ce système sera flexible et extensible, suffisamment pour les besoins de Qt 3D.

de fonctionnalités. Ces aspects interagissent avec les composants et les entités pour fournir des parties de fonctionnalités. De futurs aspects pourraient gérer la physique, l'audio, les collisions, la recherche de chemin ou encore l'intelligence artificielle.

Les prochains articles de cette série montreront la manière d'utiliser Qt 3D et l'aspect de rendu pour afficher un objet avec des *shaders* et l'animer depuis QML.

5 Article original

Le blog KDAB est rédigé par les ingénieurs de KDAB s'occupant des formations, de la consultance ainsi que du développement (de Qt et de produits additionnels). Vous pouvez trouver les versions originales : [lien 148](#).

Cet article est une traduction de l'article original

écrit par Sean Harmer paru le 16 septembre 2014 : [lien 149](#).

Cet article est une traduction de l'un des articles en anglais écrits par KDAB. Les éventuels problèmes résultant d'une mauvaise traduction ne sont pas imputables à KDAB.

Retrouvez l'article de **Sean Harmer** traduit par **Thibaut Cuvelier** en ligne : [lien 150](#)

Premiers pas avec Qt Quick-PyQt

Deuxième partie : création d'interfaces graphiques avec Qt Quick et interaction avec Python

La suite d'articles proposés a pour but d'initier le lecteur à la création d'une application écrite avec PyQt et Qt Quick.

Afin d'appréhender au mieux ce qui suit, des notions de Python et Qt Quick sont nécessaires. Vous trouverez tout ce qui peut vous être utile dans la page cours Python de Developpez.com (lien 151) et dans mon précédent article sur Qt Quick (lien 152).

De plus, une installation de PyQt 5.1 minimum est indispensable.

1 Introduction

L'article précédent (lien 153) expliquait la manière de générer une fenêtre avec des composants simples en QML dans le but de créer une première application graphique. Il est temps maintenant d'approfondir l'étude du langage QML et de découvrir la manière d'interagir entre la fenêtre et le code Python.

Les lignes de code qui suivent, dans un premier temps, auront pour objectif de créer une application graphique plus développée que celle vue dans le précédent article puis de la lancer depuis un script Python.

Dans un deuxième temps, elles montreront la manière de récupérer des valeurs de champs et aussi comment en envoyer.

L'exemple sera la création d'un simple programme d'identification d'utilisateur. Les fonction-

nalités de base attendues seront donc :

- capture d'un identifiant ;
- capture d'un mot de passe ;
- vérification des informations saisies ;
- refus ou acceptation de la connexion.

Même si son emploi n'est pas indispensable, je vous conseille fortement l'utilisation de Qt Creator. En effet, cet EDI apporte deux avantages :

- depuis sa version 2.8, il offre quelques fonctionnalités pour les développeurs Python : lien 154 ;
- il est prévu pour fonctionner avec le langage QML, base de Qt Quick (coloration syntaxique, aide à la saisie et même un Designer : lien 155).

2 Présentation de nouveaux modules Qt Quick

Le premier article montrait la création d'une interface graphique en utilisant QML et en créant chaque composant à partir de zéro : ainsi, créer un simple bouton nécessitait la création d'un rectangle, puis l'ajout de diverses propriétés en exploitant les composants de base de Qt Quick 2.

Heureusement, Qt Quick met à disposition beaucoup d'autres modules, dont certains fournissent des composants plus complets qu'un simple rectangle, par exemple.

Ces modules comptent notamment Qt Quick Controls (lien 156), qui sera entre autres l'objet du chapitre suivant.

2.1 Création d'une fenêtre et de ses composants

Qt Quick Controls contient un certain nombre de composants courants comme la fenêtre principale de

notre application, des boutons, des zones de texte, tout en restant dans la logique de Qt Quick, à savoir très personnalisables.

Dans le cadre du projet de cet article, une fenêtre de connexion, outre la fenêtre principale, il faudra une zone de saisie de l'identifiant, une zone de saisie du mot de passe, un bouton de validation et pour finir une zone de texte indiquant si la connexion est autorisée ou non.

Sans plus attendre, voici une manière d'écrire le code QML générant cette fenêtre :

```

1 import QtQuick 2.4
2 import QtQuick.Controls 1.3
3 ApplicationWindow {
4     // Création de la fenêtre principale
      avec quelques attributs
      pratiques comme menuBar et
      statusBar
5     title: qsTr("Fenêtre de connexion")
6     width: 400
7     height: 200

```

```

8   menuBar: MenuBar {
9       Menu {
10          title: qsTr("&File")
11          MenuItem {
12              text: "Une action"
13              onTriggered: console.log
                  ("Magique cette
                  barre de menu") //
                  console.log(<msg>) //
                  affiche dans la
                  console de sortie le
                  message <msg>.
14          }
15      }
16  }
17  statusBar: StatusBar {
18      Label { text: "QML est vraiment
                merveilleux" }
19  }
20  // Puis on crée et on place les
    composants souhaités.
21  Label {
22      text: "Login"
23      x: 10
24      y: 25
25  }
26  TextField {
27      id: login
28      x: 100
29      y: 20
30      placeholderText: qsTr("Enter
                your login")
31  }
32  Label {
33      text: "Mot de passe"
34      x: 10
35      y: 55
36  }
37  TextField {
38      id: password
39      x: 100
40      y: 50
41      placeholderText: qsTr("Enter
                your password")
42      echoMode: TextInput.Password
43  }
44  Button {
45      text: "Se connecter"
46      x: 10
47      y: 100
48  }
49  Label {
50      id: result
51      x: 10
52      y: 135
53  }
54 }

```

Le code proposé ne doit pas poser de problème de compréhension après le premier article. La documentation officielle des composants QML fournit plus de détails au besoin : [lien 157](#).

C'est bien joli tout ça, mais le placement des composants est loin d'être simple (positions absolues) et l'application est loin d'être très dynamique. À titre d'exemple, on ne redimensionne pas les widgets si la taille de la fenêtre change.

2.2 Premiers pas avec les dispositions

Heureusement pour nous, Qt Quick réserve encore de belles surprises comme les dispositions. Il en existe trois :

- GridLayout place ses composants dans une grille;
- ColumnLayout place ses composants en colonne;
- RowLayout place ses composants en ligne.

Ces dispositions sont fournies par le module Qt Quick Layouts. GridLayout est certes la plus complexe à utiliser, mais également la plus adaptée à la situation.

Voici le code ci-dessus retravaillé avec GridLayout :

```

1   import QtQuick 2.4
2   import QtQuick.Controls 1.3
3   import QtQuick.Layouts 1.1
4   ApplicationWindow {
5       title: qsTr("Fenêtre de connexion")
6       width: 400
7       height: 200
8       menuBar: MenuBar {
9           Menu {
10              title: qsTr("&File")
11              MenuItem {
12                  text : "Une action"
13                  onTriggered: console.log
                              ("Magique cette
                              barre de menu")
14              }
15          }
16      }
17      statusBar: StatusBar {
18          Label { text: "QML est vraiment
                    merveilleux" }
19      }
20      GridLayout {
21          id: grid
22          anchors.fill: parent // Ancre la
                              disposition au composant
                              souhaité
23          columns: 2 // Nombre
                    de colonnes
24          anchors.margins: 5 // Marge
                              entre la disposition et les
                              bords de son conteneur
25          columnSpacing: 10 // Espace
                              entre chaque colonne
26          Label {
27              text: "Identifiant"
28          }
29          TextField {
30              id: login
31              Layout.fillWidth: true
32              placeholderText: qsTr("Enter
                    your login")
33          }
34          Label {
35              text: "Mot de passe"
36          }
37          TextField {
38              id: password
39              Layout.fillWidth: true
40              placeholderText: qsTr("Enter
                    your password")
41              echoMode: TextInput.Password

```

```

42     }
43     Button {
44         text: "Se connecter"
45         // Ces deux lignes
            autorisent le
            redimensionnement
            automatique du composant
46         Layout.fillWidth: true
47         Layout.fillHeight: true
48         Layout.columnSpan: 2 // É
            talé sur deux colonnes
49     }
50     Label {
51         id: result
52         text : "Non connecté"
53         Layout.fillWidth: true
54         Layout.columnSpan: grid.
            columns // Étale sur

```

```

55     }
56 }
57 }

```

Ah... enfin un résultat simple, dynamique et très pratique. Il ne reste plus qu'à rendre le code fonctionnel en interagissant avec un code Python.



Une autre solution aurait été d'affecter un pourcentage de la hauteur et/ou de la largeur de la fenêtre à chaque composant en utilisant leurs propriétés *height* ou *width*. N'hésitez pas à faire des essais si cette solution vous intéresse.

3 Interaction avec un code Python

Maintenant que l'application est prête d'un point de vue graphique, il est temps de la rendre fonctionnelle par Python.

3.1 Lancement de la fenêtre depuis Python

Heureusement, cette intégration se fait très simplement par les lignes de code suivantes :

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import sys
4  from PyQt5.QtWidgets import QApplication
5  from PyQt5.QtQml import
    QQmlApplicationEngine
6
7  if __name__ == "__main__":
8      app = QApplication(sys.argv)
9      engine = QQmlApplicationEngine()
10     engine.load('test.qml')
11     win = engine.rootObjects()[0]
12     win.show()
13     sys.exit(app.exec_())

```

Ce script est assez simple : il crée un objet `QQmlApplicationEngine` auquel un document QML est associé et affiché. En lançant ce code, la fenêtre attendue doit apparaître.

Très bien! Après l'affichage d'une fenêtre créée en QML via Python, il reste maintenant à interagir pleinement avec elle depuis Python. Interaction avec le code.

Ce chapitre présente trois manières de créer une interaction entre du code QML et du code Python. Ces trois méthodes sont équivalentes : le choix de l'une ou l'autre se porte sur la situation ou les habitudes.



Il y aura des modifications à apporter aussi bien dans la partie Python que QML. Les éléments nouveaux seront expliqués au fur et à mesure.

3.2 Modification du code Python

Dans un premier temps, le code Python de lancement de l'application est modifié pour passer au contexte Qt Quick d'une instance de la classe `MainApp` (à créer) qui implémente la partie Python de la communication.

```

1  if __name__ == "__main__":
2      app = QApplication(sys.argv)
3      engine = QQmlApplicationEngine()
4      # Création d'un objet QQmlContext
        pour communiquer avec le code
        QML
5      ctx = engine.rootContext()
6      engine.load('test.qml')
7      win = engine.rootObjects()[0]
8      py_mainapp = MainApp(ctx, win)
9      ctx.setContextProperty("py_MainApp",
        py_mainapp)
10     win.show()
11     sys.exit(app.exec_())

```

Une fois ceci fait, il reste à créer la classe `MainApp` et à y mettre les fonctions nécessaires pour la communication. La première méthode consiste à renvoyer la réponse par une propriété du contexte global : pour y accéder en QML, il suffira de taper le nom de la propriété.

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  import sys
4  from PyQt5.QtCore import QObject,
    pyqtSlot, QVariant
5
6  # Classe servant dans l'interaction.
7  class MainApp(QObject):
8      def __init__(self, context, parent=
        None):
9          super(MainApp, self).__init__(
        parent)
10     # Recherche d'un enfant appelé
        myButton dont le signal
        clicked sera connecté à la
        fonction test3
11     self.win = parent
12     self.win.findChild(QObject, "
        myButton").clicked.connect(

```

```

13         self.test3)
14         self.ctx = context
15     # Création de la fonction d'
16     # identification
17     def verifConnection(self, login,
18     password):#voir PEP8
19     if login == "login" and password
20     == "mdp":
21         return "Bonjour %s. Vous ê
22         tes maintenant connecté"
23         % login
24     else:
25         return "Désolé,
26         authentification
27         impossible"
28
29     # Premier test de communication :
30     # propriétés contextuelles.
31     @pyqtSlot(QVariant, QVariant)
32     def test1(self, login, password):
33         txt = self.verifConnection(login
34         , password)
35     # Transmission du résultat comme
36     # une propriété nommée retour
37     self.ctx.setTextProperty("
38     retour", txt)
39     return 0
40
41     # Deuxième test de communication :
42     # création d'une fonction ayant
43     # pour type de retour un QVariant
44     # (obligatoire ici pour que QML
45     # sache l'interpréter).
46     @pyqtSlot(QVariant, QVariant, result
47     =QVariant)
48     def test2(self, login, password):
49         return self.verifConnection(
50         login, password)
51
52     # Troisième test de communication :
53     # modification directe d'un
54     # composant QML.
55     def test3(self):
56     # Recherche des enfants par leur
57     # attribut objectName, récupé
58     # ration de la valeur de leur
59     # propriété text
60     login = self.win.findChild(
61     QObject, "myLogin").property
62     ("text")
63     password = self.win.findChild(
64     QObject, "myPassword").
65     property("text")
66     txt = self.verifConnection(login
67     , password)
68     self.win.findChild(QObject, "
69     labelCo").setProperty("text"
70     , txt)
71     return 0

```

3.3 Modification du code QML

Maintenant que notre code Python est prêt, il ne reste plus qu'à modifier légèrement le code QML pour exploiter ces moyens de communication.

```

1 import QtQuick 2.4
2 import QtQuick.Controls 1.3
3 import QtQuick.Layouts 1.1
4 ApplicationWindow {
5     title: qsTr("Fenêtre de connexion")

```

```

6     width: 400
7     height: 200
8     menuBar: MenuBar {
9         Menu {
10            title: qsTr("&File")
11            MenuItem {
12                text: "Une action"
13                onTriggered: console.log
14                ("Magique cette
15                barre de menu")
16            }
17        }
18    }
19    statusBar: StatusBar {
20        Label { text: "QML est vraiment
21        merveilleux" }
22    }
23    GridLayout {
24        id: grid
25        anchors.fill: parent
26        columns: 2
27        anchors.margins: 5
28        columnSpacing: 10
29        Label {
30            text: "Identifiant"
31        }
32        TextField {
33            id: login
34            objectName: "myLogin" //
35            test3() cherche un enfant
36            nommé myLogin
37            Layout.fillWidth: true
38            placeholderText: qsTr("Enter
39            your login")
40        }
41        Label {
42            text: "Mot de passe"
43        }
44        TextField {
45            id: password
46            objectName: "myPassword"
47            Layout.fillWidth: true
48            placeholderText: qsTr("Enter
49            your password")
50            echoMode: TextInput.Password
51        }
52        Button {
53            text: "Se connecter par test
54            1()"
55            Layout.fillWidth: true
56            Layout.fillHeight: true
57            Layout.columnSpan: 2
58            onClicked: {
59                py_MainApp.test1(login.
60                text, password.text)
61                result.text = retour
62            }
63        }
64        Button {
65            text: "Se connecter par test
66            2()"
67            Layout.fillWidth: true
68            Layout.fillHeight: true
69            Layout.columnSpan: 2
70            onClicked: result.text =
71            py_MainApp.test2(login.
72            text, password.text)
73        }
74        Button {
75            text: "Se connecter par test
76            3()"
77            objectName: "myButton"
78            Layout.fillWidth: true

```



```
66         Layout.fillHeight: true
67         Layout.columnSpan: 2
68     }
69     Label {
70         id: result
71         objectName: "labelCo"
72         text : "Non connecté"
```

```
73         Layout.fillWidth: true
74         Layout.columnSpan: grid.
75             columns
76     }
77 }
```

4 Conclusion

Après une première partie destinée à découvrir Qt Quick, cette deuxième montre comment créer une interface graphique très fonctionnelle avec un minimum de code ainsi que l'interaction entre du code Python et du code QML.

La lecture de ces deux premières parties doit vous permettre d'imaginer sereinement vos prochaines lignes de code. Cependant, Qt Quick possède encore beaucoup d'autres fonctionnalités qui seront abordées dans la troisième et dernière partie.

Retrouvez l'article de *Charlie Gentil* en ligne : [lien 158](#)

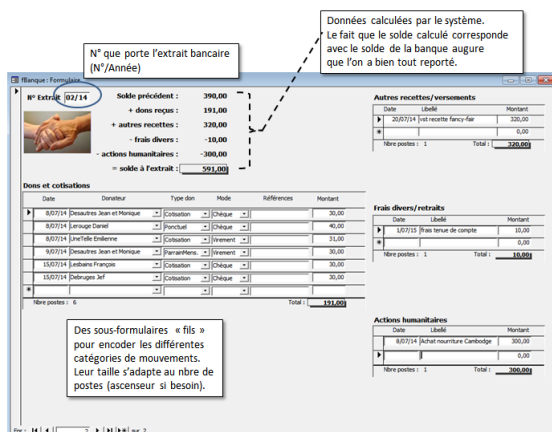
3.2 fBanque et fCaisse



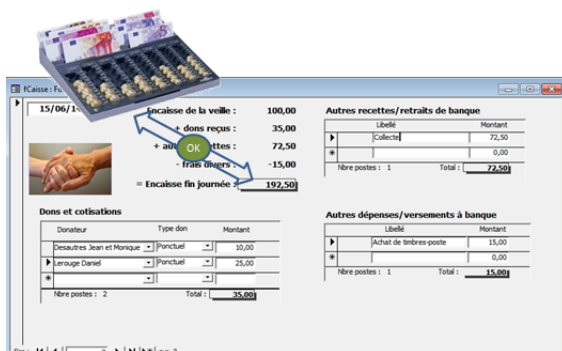
L'idée est de concevoir des formulaires :

- qui permettent de saisir les entrées et sorties de fonds en les classant par nature (cotisations reçues, octrois d'aide financière, frais de fonctionnement...) comme dans une comptabilité en partie double ;

- qui incluent un système d'autocontrôle. Par exemple, si le formulaire *fbanque* restitue le cumul des entrées et sorties qui y sont enregistrées et que cette somme correspond au solde renseigné à l'extrait délivré par la banque, il y a de fortes chances que l'encodage soit complet.

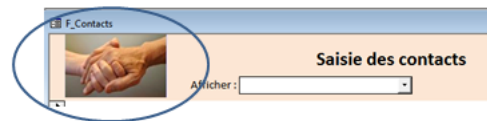


De même pour les espèces en caisse



3.3 Quelques commentaires techniques

3.3.a Les images ne sont pas dans la base de données



Elles sont logées dans un sous-répertoire IMAGES.

À l'ouverture du formulaire, ce code

```
1 Private Sub Form_Open(Cancel As Integer)
2 Call AmnImages
3 End Sub
```

La routine appelée est logée dans le module *mImages*.

Le principe est le suivant :

- le contrôle image a cette structure

- à l'ouverture du formulaire (ou de l'état), la routine *AmnImages* repère tous les contrôles de type *Image* (Picture) en modifiant le chemin de l'image à afficher

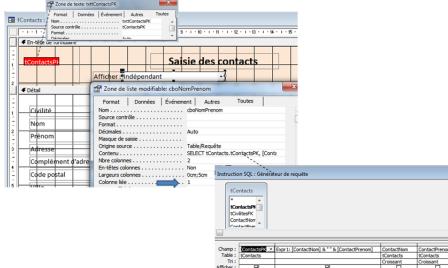
```
1 For Each Ctrl In CodeContextObject.
   Controls
2     If Ctrl.Picture = "(aucune)" And
       Ctrl.PictureType = 1 And Ctrl.
       Tag Like "*.*)" Then
3         Ctrl.Picture = CurrentProject.Path
           & "\Images\" & Ctrl.Tag
4     End If
5 Next Ctrl
```

Vous trouverez plus de détails sur cette routine dans cet autre tutoriel : « Stockez les images statiques de vos formulaires et états Access hors de la base de données » (lien 164).

3.3.b Se positionner sur un enregistrement précis



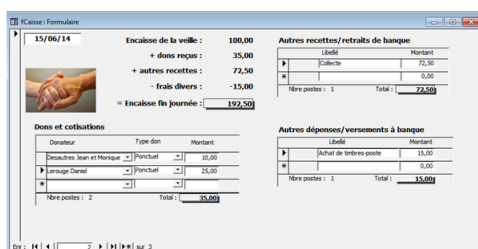
Quand l'utilisateur a choisi un contact dans la liste, le contrôle `cboNomPrenom` contient la clé de ce contact `tContactsPK`



Pour se positionner sur l'enregistrement choisi, remettre la zone de liste modifiable à Null et donner le focus sur le nom et prénom :

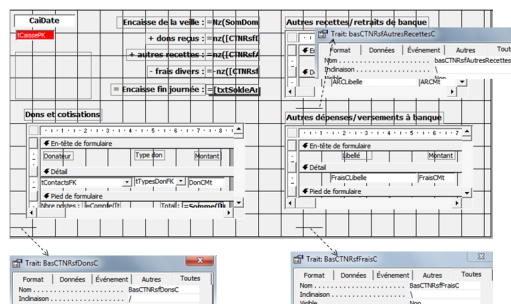
```
1 Private Sub cboNomPrenom_AfterUpdate ()
2   DoCmd.GoToControl "txttContactsPK"
3   DoCmd.FindRecord Me.cboNomPrenom
4   Me.cboNomPrenom = Null
5   Me.txtContactNom.SetFocus
6 End Sub
```

3.3.c La hauteur des sous-formulaires s'adapte suivant le nombre d'enregistrements



Ceci pour que la ligne « Total » colle à la dernière ligne du détail.

En bref, la hauteur du conteneur dépend du nombre enregistrements du formulaire fils. On place dans le formulaire « père » une limite au-delà de laquelle on veut stopper la progression. Une barre de défilement verticale se met alors en place.



Le processus d'adaptation est déclenché à chaque affichage d'un enregistrement du père

```
1 Private Sub Form_Current ()
2   'Moduler la taille des sous-
3   formulaires
4   Call AmenagerTailleSF(Me.Name, "
5   CTNRsfDonsC")
6   Call AmenagerTailleSF(Me.Name, "
7   CTNRsfAutresRecettesC")
8   Call AmenagerTailleSF(Me.Name, "
9   CTNRsfFraisC")
10  Me.txtCaiDate.SetFocus
11 End Sub
```

et à chaque ajout ou suppression d'un enregistrement dans le fils

```
1 Private Sub Form_AfterDelConfirm(Status
2   As Integer)
3   Call AmenagerTailleSF(Me.Parent.Name,
4   Me.Parent.ActiveControl.Name)
5   Me.Requery
6 End Sub
7
8 Private Sub Form_AfterInsert ()
9   Call AmenagerTailleSF(Me.Parent.Name,
10  Me.Parent.ActiveControl.Name)
11 Me.Requery
12 End Sub
```

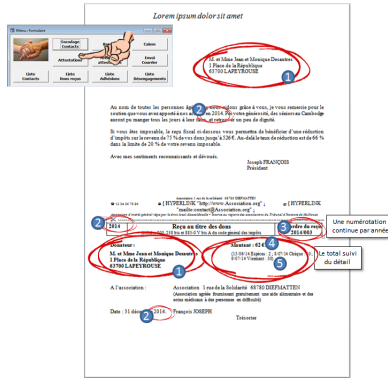
La procédure `AmenagerTailleSF ()` est logée dans le module `mTailleSF` .

Un commentaire détaillé de ce code peut être consulté dans ce tutoriel : [lien 165](#).

4 Confection des attestations

4.1 Le défi

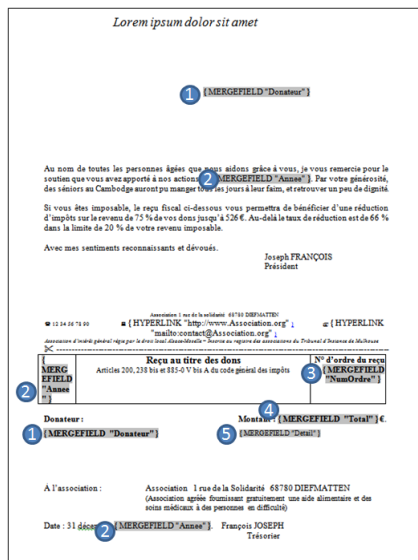
En un clic sur un bouton, les attestations fiscales de l'année sont confectionnées, - sur papier pour chaque donateur sans adresse e-mail; - sous forme d'un PDF, joint à un mail pour les autres.




Nous allons piloter un publipostage Word depuis Access.

4.2 Le modèle de base

Voici comment il se présentera :

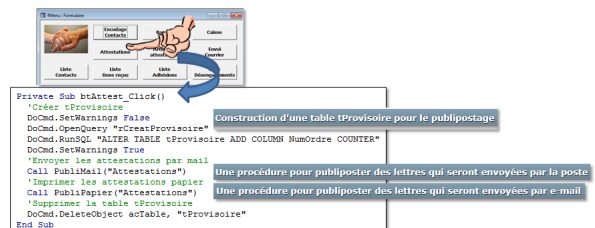


 Si la technique ne vous est pas familière, voyez une description dans ce tutoriel : lien 166

Nous devons donc construire une table qui contiendra pour chaque donateur :

Nom de la colonne	Contenu
1 Donateur	4 lignes Civilité, prénom, nom Adresse Complément d'adresse éventuel Code postal, localité
2 Annee	L'année des dons
3 NumOrdre	Une numérotation continue
4 Total	Le total des dons
5 Detail	Pour chaque don La date, le mode, le montant

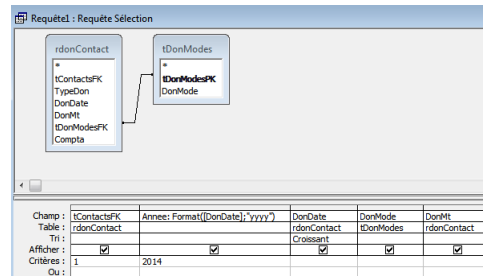
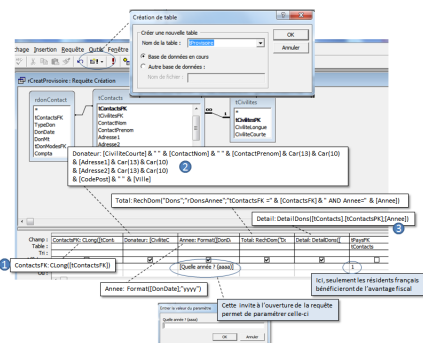
4.3 Un petit clic, mais un paquet de code



4.4 Construction d'une table tProvisoire pour le publipostage

4.4.a La requête rCreatProvisoire

```
1 SELECT DISTINCT CLng([tContactsFK]) AS
   ContactsFK, [CivilliteCourte] & " " & [ContactNom] & " " & [ContactPrenom]
   & Chr(13) & Chr(10) & [Adresse1] &
   Chr(13) & Chr(10) & [Adresse2] &
   Chr(13) & Chr(10) & [CodePost] & " "
   & [Ville] AS Donateur, Format([
   DonDate], "yyyy") AS Annee, DLookup("
   Dons", "rDonsAnnee", "tContactsFK =" &
   [ContactsFK] & " AND Annee=" & [
   Annee]) AS Total, DetailDons([
   tContacts].[tContactsPK],[Annee]) AS
   Detail INTO tProvisoire
2 FROM tCivilites INNER JOIN (rdonContact
   INNER JOIN tContacts ON rdonContact.
   tContactsFK = tContacts.tContactsPK)
   ON tCivilites.tCivilitesPK =
   tContacts.tCivilitesFK
3 WHERE (((Format([DonDate], "yyyy"))=[
   Quelle année ? (aaaa)]) AND ((
   tContacts.tPaysFK)=1));
```



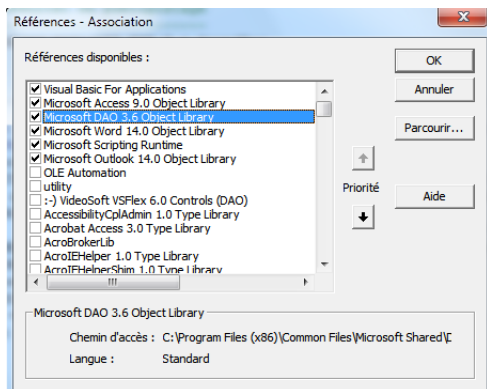
- 1 Cette transformation en Long pour éviter que la colonne créée soit NuméroAuto.
- 2 & Car(13) & Car(10) provoque un saut à la ligne lors de l'impression.
- 3 La fonction *Detail()*

```

1 Public Function DetailDons(tContactsPK
  As Long, Annee As Integer) As String
2 Dim sSql As String
3 Dim rs As Recordset
4 sSql = "SELECT rdonContact.tContactsFK
  , Format([DonDate], "yyyy") AS
  Annee, " -
5 & "rdonContact.DonDate, tDonModes.
  DonMode, rdonContact.DonMt " -
6 & "FROM rdonContact INNER JOIN tDonModes
  " -
7 & "ON rdonContact.tDonModesFK =
  tDonModes.tDonModesPK " -
8 & "WHERE tContactsFK =" & tContactsPK &
  " And Format([DonDate], "yyyy") ="
  & Annee -
9 & " ORDER BY rdonContact.DonDate;"
10 Set rs = CurrentDb.OpenRecordset(sSql)
11 Do Until rs.EOF
12 DetailDons = DetailDons & rs("
  DonDate") & " " & rs("DonMode")
  & " : " & Format(rs("DonMt"), "
  #.00") & " "
13 rs.MoveNext
14 Loop
15 End Function
  
```

Explication du code

1-2 : la définition d'une variable de type Recordset implique d'ajouter la bibliothèque Microsoft DAO x.x Object Library au projet



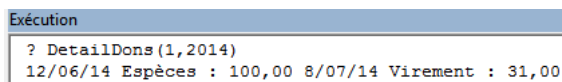
4-9 : on construit le SQL d'une requête qui aurait cet aspect si on appelait la fonction comme ceci DetailDons(1,2014)

donc une requête qui ramène tous les dons reçus de la part du donateur 1 pour l'année 2014.

10 : on crée un jeu d'enregistrements (recordset) avec ce SQL.

11-14 : on lit un à un les enregistrements et pour chacun (12), on concatène la date, le mode et le montant.

En l'occurrence, la fonction donnerait ceci :



4.4.b On ajoute une colonne NuméroAuto à la table tProvisoire

Avec ce code

```

1 DoCmd.RunSQL "ALTER TABLE tProvisoire
  ADD COLUMN NumOrdre COUNTER"
  
```

4.4.c À ce stade, voici à quoi ressemble tProvisoire

ContactsFK	Donateur	Annee	Total	Detail	NumOrdre
3	M. et Mme Lesbains Fr	2014	60,55	20/06/14 Espéc	1
4	M. et Mme Mouton Ben	2014	300	1/07/14 Chèque	2
5	M. et Mme Desautres J	2014	70	15/06/14 Espéc	3
6	M. et Mme Lerouge Dar	2014	65	15/06/14 Espéc	4

4.5 Une procédure pour publier les lettres qui seront mises sous pli

4.5.a Les étapes



4.5.b Le code

```

1 Public Sub PubliPapier(Genre As String)
2   Dim objWord As Word.Application
3   Dim NomDoc As String
4   Dim ModeleDoc As String
5   Dim SavedDoc As String
6
7   'Ajuster les paramètres en fonction du
8   Genre
9
10  Select Case Genre
11
12    Case "Attestations"
13      ModeleDoc = CurrentProject.Path &
14        "\ATTESTATIONS\
15        AttestationsModele.doc"
16      SavedDoc = CurrentProject.Path & "
17        \AEnvoyer\AttestationsAEnvoyer
18        .doc"
19      'Créer la table tPubliPapier
20      DoCmd.SetWarnings False
21      DoCmd.RunSQL "SELECT tProvisoire.*
22        , tContacts.Courriel1 INTO
23        tPubliPapier " _
24      & "FROM tContacts INNER JOIN tProvisoire
25        " _
26      & "ON tContacts.tContactsPK =
27        tProvisoire.ContactsFK " _
28      & "WHERE tContacts.Courriel1 Is Null;"
29      DoCmd.SetWarnings True
30
31    Case "ArchivAttest"
32      DoCmd.SetWarnings False
33      DoCmd.RunSQL "SELECT tProvisoire.*
34        , tContacts.Courriel1 INTO
35        tPublipapier " _
36      & "FROM tContacts INNER JOIN tProvisoire
37        " _
38      & "ON tContacts.tContactsPK =
39        tProvisoire.ContactsFK;"
40      DoCmd.SetWarnings True
41      ModeleDoc = CurrentProject.Path &
42        "\ATTESTATIONS\
43        AttestationsModele.doc"
44      SavedDoc = CurrentProject.Path &
45        "\ATTESTATIONS\Archives\" _
46      & DLookup("Annee", "tPubliPapier") & "
47        ArchivageAttestations.doc"
48
49    Case "Courrier"
50      ModeleDoc = Forms!fEnvoiCourrier!
51        txtCheminDoc
52      SavedDoc = CurrentProject.Path &
53        "\AEnvoyer\CourrierAEnvoyer.
54        doc"
55      DoCmd.SetWarnings False
56      DoCmd.RunSQL "SELECT * INTO
57        tPubliPapier " _
58      & "FROM tCourrier WHERE tCourrier.Poste=
59        True;"
60      DoCmd.SetWarnings True
61
62  End Select
63
64  'Publiposter
65  '-----
66  Set objWord = New Word.Application
67  With objWord
68    Ouvrir le document modèle
69    .Documents.Open ModeleDoc

```

```

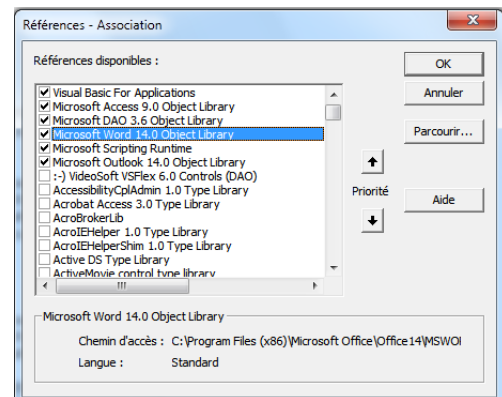
49   .ActiveDocument.MailMerge.
50     OpenDataSource _
51       Name:=CurrentDb.Name,
52       SQLStatement:="SELECT *
53         FROM [tPubliPapier]"
54   .ActiveDocument.MailMerge.Execute
55   .ActiveDocument.SaveAs2 SavedDoc
56   .Documents.Close
57 End With
58
59 ' Fermer et libérer les objets
60 '-----
61 objWord.Quit
62 Set objWord = Nothing
63
64 'Supprimer la table tPubliPapier
65 '-----
66 DoCmd.DeleteObject acTable, "
67   tPubliPapier"
68
69 'Montrer le résultat du publipostage
70 '-----
71 Shell "C:\WINDOWS\EXPLORER.EXE " &
72   SavedDoc
73 End Sub

```

Commentaire du code

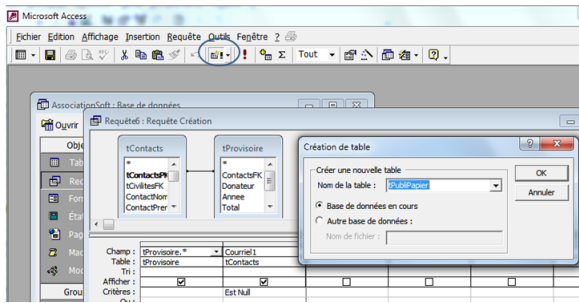
1 : cette routine contient un paramètre qui renseigne le genre de document à publier. Ici, il s'agira de « Attestations », d'autres cas seront évoqués plus loin).

2-5 : on définit des variables, dont Dim objWord As Word.Application qui implique que l'on ajoute la bibliothèque *Microsoft Word xx.x Object Library* au projet (« xx.x » correspond à la version installée sur votre machine)



10-41 : on ajuste le contenu de certaines variables en fonction du paramètre. 13 : dans notre cas, le document modèle décrit plus haut a été logé dans le sous-répertoire « ATTESTATIONS » et s'appelle « AttestationsModele.doc ».

14 : le document issu du publipostage sera logé dans le sous-répertoire « AEnvoyer » et s'appellera « AttestationsAEnvoyer.doc ». 17-20 : on crée une table *tPubliPapier*, le SQL correspond à cette requête :



En clair, on prend dans *tProvisoire* tous les contacts qui n'ont pas d'adresse e-mail.

22-41 : ces instructions concernent d'autres valeurs du paramètre.

43-54 : le publipostage proprement dit : 45 : on interface Access avec Word ; 48-53 : on déclenche la procédure de publipostage dans Word, avec le document *ModeleDoc* et on loge le résultat à l'adresse

SavedDoc.

58-59 : on referme Word et on libère la mémoire.

63 : on supprime la table *tPubliPapier* désormais inutile.

67 : on ouvre le document à publiposter : l'utilisateur n'a plus qu'à l'imprimer et à le mettre sous enveloppe.

Remarquez cette syntaxe « passe-partout »

```
Shell "C:\WINDOWS\EXPLORER.
EXE " &
LeCheminDunFichier
```



Cela correspond à un double-clic sur le fichier quand vous êtes dans Explorer : le fichier s'ouvre avec le programme qui lui est associé.

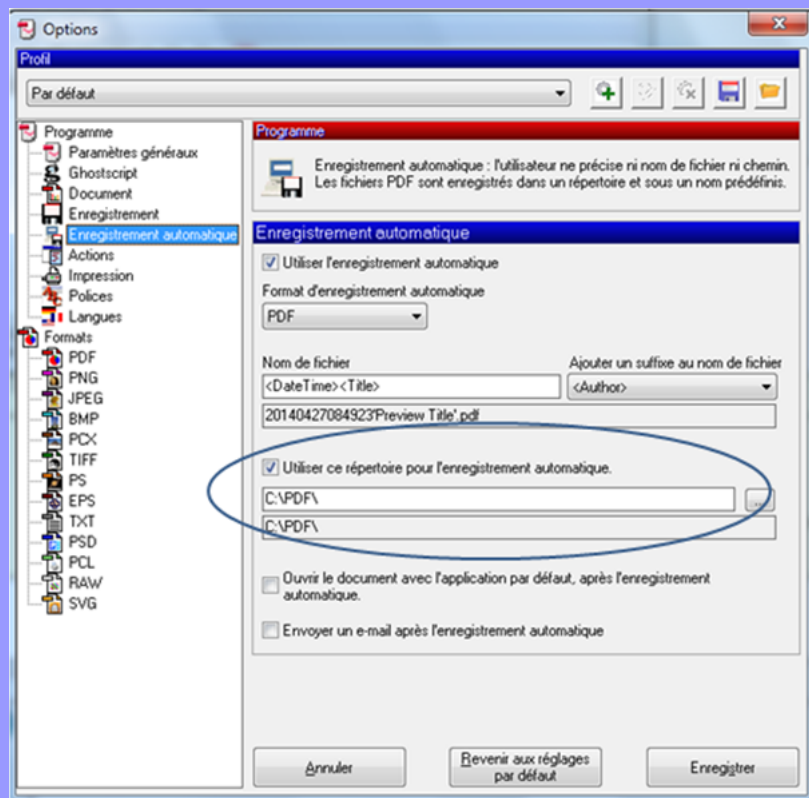
4.6 Une procédure pour publiposter des lettres qui seront envoyées par e-mail

Pour utiliser le code suivant, vous devez disposer de PDFCreator, c'est un programme gratuit qui vous permet d'installer une imprimante qui, au lieu de produire du papier, crée un fichier .pdf.

Chercher sur internet un endroit pour le télécharger.

Voici les paramètres qui ont été choisis pour l'installation : l'enregistrement automatique dans le répertoire C : \PDF\ (Vous pouvez évidemment choisir une autre localisation et adapter la suite du code en conséquence. Seule précaution, le répertoire choisi ne doit rien contenir d'autre, car nous le vidangeons à chaque envoi.)

Menu Imprimante > Options > Enregistrement automatique




```

66 & "FROM tPubliMail WHERE ContactsFK=" &
    rst("ContactsFK") & ";"
67     Case "Courrier"
68         DoCmd.RunSQL "SELECT * INTO
        tPubliUnMail " _
69 & "FROM tPubliMail WHERE NomPrenom="" &
    rst("NomPrenom") & "";"
70     End Select
71     DoCmd.SetWarnings True
72
73     'Créer le PDF de cette attestation
74     '-----
75     'S'assurer que c:/pdf est vide
76     Set oFSO = New Scripting.
        FileSystemObject
77     Set oFld = oFSO.GetFolder("c:\pdf")
78     For Each oFl In oFld.Files
79         oFl.Delete
80     Next oFl
81
82     'Publiposter
83     '-----
84     With objWord
85         .Visible = True
86         ' Ouvrir le document type
87         .Documents.Open ModeleDoc
88         .ActiveDocument.MailMerge.
            OpenDataSource _
89             Name:=CurrentDb.Name,
                SQLStatement:="SELECT *
                FROM [tPubliUnMail]"
90         .ActiveDocument.MailMerge.Execute
91         .ActiveDocument.PrintOut
92         .Documents.Close SaveChanges:=
            wdDoNotSaveChanges
93     End With
94     Sleep 2000 'pause de 2 s pour créer
        le document
95
96     'Renommer le fichier PDF
97     '-----
98     For Each oFl In oFld.Files
99         oFl.Name = NomPDF
100     Next
101
102     'Expédier le mail avec le PDF en pié
        ce jointe
103     '-----
104
105     'Composer le message
106     Set MonMessage = objOutlook.
        createitem(0) 'ouvrir une
        structure de message
107     MonMessage.To = rst("Courriel1")
108     MonMessage.Subject = ObjetMail
109     MonMessage.Body = MessageMail
110     MonMessage.Attachments.Add "c:\pdf\"
        & NomPDF
111     MonMessage.send
112     Sleep 2000 'pause de 2 s pour l'
        envoi
113
114     'Au suivant...
115     rst.MoveNext
116 Loop
117 Sortie:
118
119     'Rétablir l'imprimante par défaut
120     wsn.SetDefaultPrinter sImprDefaut
121     Set wsn = Nothing
122
123     'Fermer Outlook et Word

```

```

124     objOutlook.Quit
125     Set objOutlook = Nothing
126     objWord.Quit
127     Set objWord = Nothing
128
129     'Libérer le recordset
130     rst.Close
131     Set rst = Nothing
132
133     'Supprimer les tables temporaires
134     DoCmd.DeleteObject acTable, "
        tPubliMail"
135     DoCmd.DeleteObject acTable, "
        tPubliUnMail"
136
137 End Sub

```

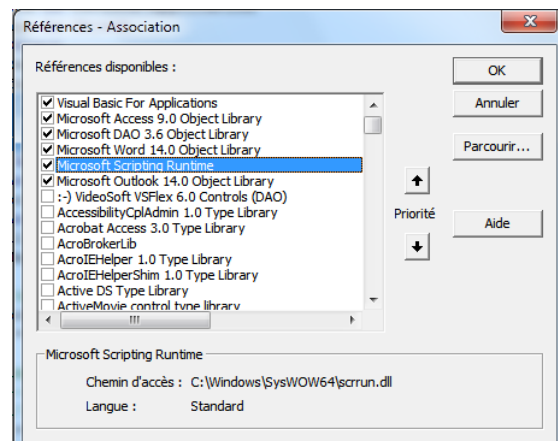
Commentaire du code

Cette routine contient un paramètre qui renseigne le genre de document à publiposter. Ici, il s'agira de « Attestations », d'autres cas seront évoqués plus loin.

2-15 : on définit des variables dont

- Dim objOutlook As Outlook.Application qui implique que l'on ajoute la bibliothèque **Microsoft Outlook xx.x** au projet (« xx.x » correspond à la version installée sur votre machine);

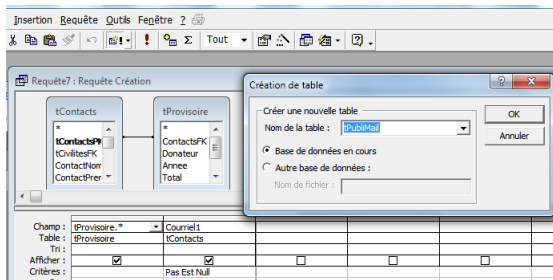
- Dim oFSO As Scripting.FileSystemObject, Dim oFld As Folder et Dim oFl As File qui impliquent **Microsoft Scripting Runtime**.



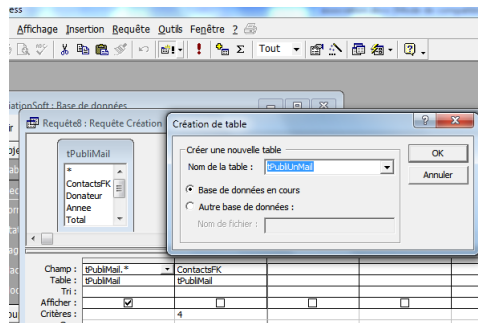
22-27 : on mémorise les coordonnées de l'imprimante actuellement par défaut (pour pouvoir la rétablir *in fine*) et on installe PDFCreator à la place.

24 : la fonction `ImprimanteParDefaut()` se trouve dans le module **mDivers**. (Désolé pour le manque d'explication, c'est un copier-coller de quelque part. Je n'ai pas tout compris, mais ça marche!)


40-43 : on crée une table **tPubliMail** qui contient tous les contacts à qui envoyer une attestation par e-mail. Le SQL généré correspond à ceci :



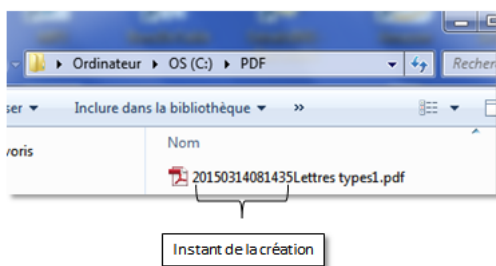
65-66 : dans la boucle de lecture des enregistrements de *tPubliMail*, on crée une table *tPubliUnMail* qui ne contient que le seul enregistrement en cours de lecture.



75-80 : on vidange le répertoire *c:\pdf*. Ce sera plus simple pour rebaptiser le fichier PDF que nous allons créer par publipostage.

 Il est donc impératif que le répertoire choisi pour loger les PDF générés par PDFCreator (dans l'exemple : *c:\pdf*) ne contienne aucun fichier que vous voudriez garder !

91 : le résultat du publipostage sera logé dans le répertoire *c:\pdf* et portera le nom que PDFCreator lui aura automatiquement donné. Quelque chose comme ceci :



5 Archivage des attestations fiscales

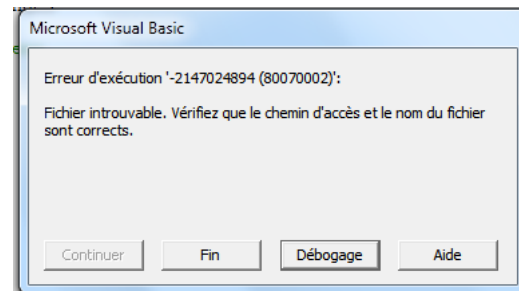
Une copie des attestations doit être sauvegardée en vue d'un éventuel contrôle du fisc.

La procédure est une variante de ce que nous venons de décrire au chapitre précédent.

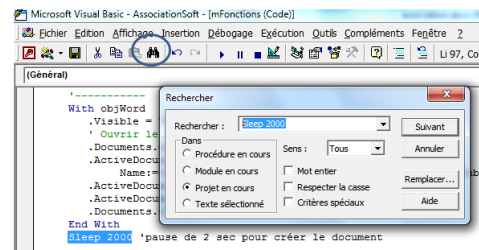
Nous publipostons sur papier pour l'ensemble des contacts concernés (qu'ils aient ou non une adresse e-mail) :

```
1 Private Sub btArchiAttest_Click()
```

94 : on fait une pause de deux secondes pour laisser le temps à Word de s'exécuter. En effet Access fonctionne de manière asynchrone, c'est-à-dire que dans le code une instruction n'attend pas que la précédente soit terminée. En l'occurrence, les instructions 98-100 qui vont rebaptiser le fichier PDF pourraient s'exécuter avant que le fichier PDF ne soit créé ! Ceci léverait ce message d'erreur



Le délai nécessaire (ici 2 s) dépend de la vitesse de votre machine. Ajustez si nécessaire toutes les occurrences de Sleep dans tout le code du projet



98-100 : on rebaptise le fichier .pdf en lui donnant le nom « Attestation.pdf » (attribué à la variable *NomPDF* à l'instruction 35.

106-112 : on construit l'e-mail et on l'expédie avec Attestation.pdf en pièce jointe.

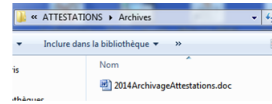
La suite du code n'appelle pas davantage de commentaires.

```
2 'Créer tProvisoire
3 DoCmd.SetWarnings False
4 DoCmd.OpenQuery "rCreatProvisoire"
5 DoCmd.RunSQL "ALTER TABLE tProvisoire
6 ADD COLUMN NumOrdre COUNTER"
7 DoCmd.SetWarnings True
8 'Créer le document à archiver
9 Call PubliPapier("ArchivAttest")
10 'Supprimer la table tProvisoire
DoCmd.DeleteObject acTable, "
```

```
11 End Sub tProvisoire"
```

```
1. Public Sub PubliPapier (Genre As String)
23. Case "ArchivAttest"
24. DoCmd.SetWarnings False
25. DoCmd.RunSQL "SELECT tProvisoire.*, tContacts.Courriell INTO tPubliPapier "
26. & "FROM tContacts INNER JOIN tProvisoire "
27. & "ON tContacts.tContactsFK = tProvisoire.ContactsFK;"
28. DoCmd.SetWarnings True
29. ModelDoc = CurrentProject.Path & "\ATTESTATIONS\AttestationsModele.doc"
30. SavedDoc = CurrentProject.Path & "\ATTESTATIONS\Archives\"
31. & DLookup("Annee", "tPubliPapier") & "ArchivageAttestations.doc"
32.
```

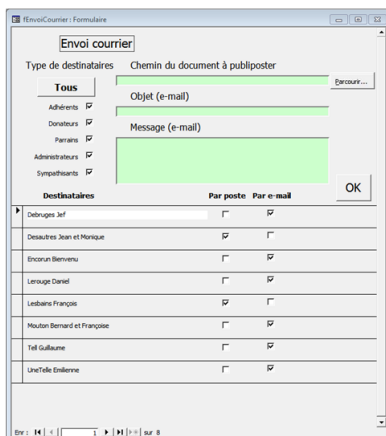
- 1 On prend tous les contacts qui ont eu une attestation
- 2 On sauvegarde le résultat de publipostage dans le sous-sous-répertoire « Archives »



6 Envoi de courrier ciblé

6.1 Le formulaire fEnvoiCourrier

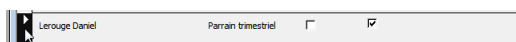
À l'ouverture



- Tous les types de contacts sont sélectionnés comme destinataires. (Sans doublon pour les contacts qui ont plusieurs types d'engagements.)
- Pour éliminer des types de contacts, il suffit de décocher les cases non désirées

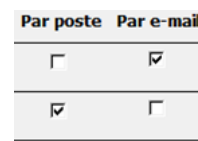


- On peut aussi supprimer des enregistrements de manière manuelle, en les sélectionnant



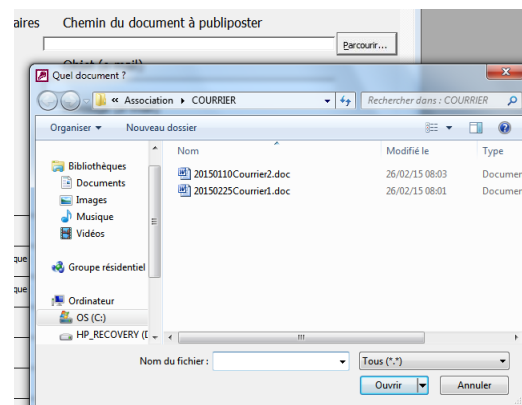
et en pressant la touche ou <Suppr>.

- Selon que le contact a donné une adresse courriel ou non, la case est cochée par défaut



(Il est possible de modifier ponctuellement la voie d'expédition pour autant qu'elle soit disponible pour ce contact.)

- Pour renseigner le chemin du document à publiposter, on peut :
 - soit l'introduire manuellement ;
 - soit cliquer le bouton Parcourir... qui ouvre une fenêtre de recherche de fichier.



- Cliquer le bouton OK déclenche le processus d'envoi.

6.2 Le code

```
1 Option Compare Database
2 Option Explicit
3
4 Private Sub Form_Open (Cancel As Integer)
5 Me.RecordSource = ""
6 DoCmd.SetWarnings False
7 DoCmd.OpenQuery "rCreaCourrier"
8 DoCmd.SetWarnings True
9 Me.RecordSource = "tCourrier"
10 End Sub
11
```

```

12 Private Sub Form_Unload(Cancel As
    Integer)
13     Me.RecordSource = ""
14     DoCmd.DeleteObject acTable, "tCourrier"
15 End Sub
16
17 Private Sub BtChercher_Click()
18     Me.txtCheminDoc = OuvrirUnFichier(Me.
        hwnd, "Quel document ?", 1, , ,
        CurrentProject.Path & "\COURRIER")
19 End Sub
20
21 Private Sub btOK_Click()
22     Dim oFSO As Scripting.FileSystemObject
23     Me.Refresh
24     'Vérifier complétude du formulaire
25     If IsNull(Me.txtCheminDoc) + IsNull(Me.
        txtMessMail) + IsNull(Me.txtObjet
        ) <> 0 Then
26         MsgBox "Des champs obligatoires
            manquent"
27         Exit Sub
28     End If
29     'Vérifier la présence du modèle
30     Set oFSO = New Scripting.
        FileSystemObject
31     If oFSO.FileExists(Me.txtCheminDoc) =
        False Then
32         MsgBox "Le document à publier
            est introuvable", vbCritical
33     Me.txtCheminDoc.SetFocus
34     Exit Sub
35     End If
36     'Créer les lettres si demandées
37     If DSum("Poste", "tCourrier") <> 0
        Then Call PubliPapier("Courrier")
38     'Créer les e-mail si demandés
39     If DSum("[e-mail]", "tCourrier") <> 0
        Then Call PubliMail("Courrier")
40 End Sub
41
42 Private Sub BtTous_Click()
43     Dim ctl As Control
44     For Each ctl In Me.Controls
45         If ctl.Name Like "cc*" Then
46             ctl = -1
47         End If
48     Next ctl
49     Me.Repaint
50     Call Form_Open(0)
51 End Sub
52
53 Private Sub ccAdmi_AfterUpdate()
54     Call Form_Open(0)
55 End Sub
56
57 Private Sub ccDona_AfterUpdate()
58     Call Form_Open(0)
59 End Sub
60
61 Private Sub ccParr_AfterUpdate()
62     Call Form_Open(0)
63 End Sub
64
65 Private Sub ccSymp_AfterUpdate()
66     Call Form_Open(0)
67 End Sub
68
69 Private Sub ccAdhe_AfterUpdate()
70     Call Form_Open(0)
71 End Sub
72

```

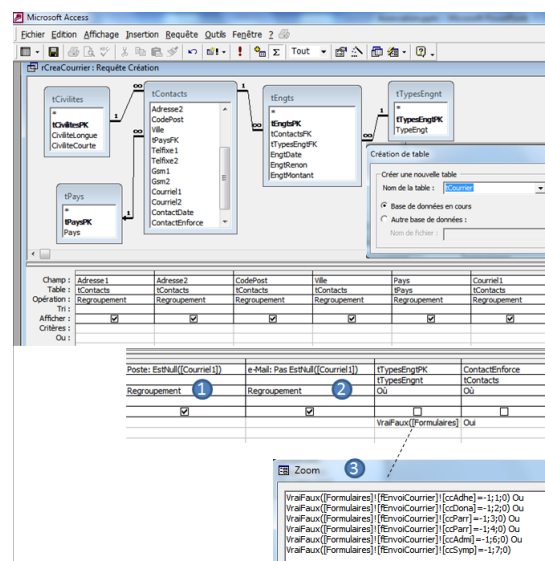
```

73 Private Sub Poste_BeforeUpdate(Cancel As
    Integer)
74
75     If Me.Poste = True Then
76         If IsNull(DLookup("Adresse1", "
            tCourrier", "NomPrenom=" & Me.
            txtNomPrenom & """)) Then
77             MsgBox "Ce contact n'a pas d'
                adresse postale", vbCritical
78             Cancel = True
79             Me.Poste.Undo
80         End If
81     End If
82 End Sub
83
84 Private Sub e_Mail_BeforeUpdate(Cancel
    As Integer)
85     If Me.e_Mail = True Then
86         If IsNull(DLookup("Courriel1", "
            tCourrier", "NomPrenom=" & Me.
            txtNomPrenom & """)) Then
87             MsgBox "Ce contact n'a pas d'
                adresse e-mail", vbCritical
88             Cancel = True
89             Me.e_Mail.Undo
90         End If
91     End If
92 End Sub

```

Commentaire du code

4-10 : à l'ouverture, on exécute la requête *rCreaCourrier* qui crée (provisoirement) une nouvelle table *tCourrier*.



- 1 la valeur de la colonne sera égale à -1 si le contact n'a pas d'adresse e-mail, 0 s'il en a une.
- 2 le contraire : la colonne sera égale à -1 si le contact a une adresse e-mail, 0 s'il n'en a pas.
- 3 seulement pour les tTypesEngtPK qui correspondent aux cases cochées dans le formulaire. (À l'ouverture, elles sont toutes cochées par défaut.)

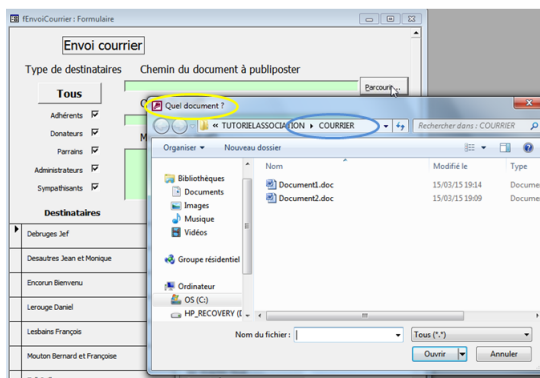
tTypesEngtPKLa table tCourrier ainsi créée :

Nom du champ	Type de données
CiviliteCourte	Texte
NomPrenom	Texte
Adresse1	Texte
Adresse2	Texte
CodePost	Texte
Ville	Texte
Pays	Texte
Courriel1	Texte
Poste	Numérique
e-Mail	Numérique

Cette table est affectée comme source au formulaire (instruction 9).

12-15 : la table sera supprimée lors de la fermeture du formulaire.

17-19 : un clic sur le bouton « Chercher » déclenche l'ouverture d'une boîte de dialogue pour le choix d'un fichier :

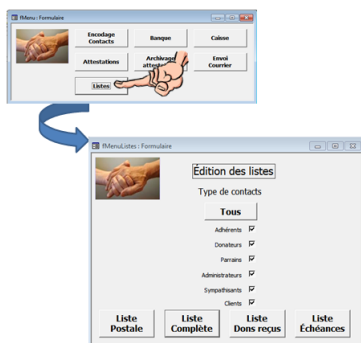


Le DialogBox (lien 167) a été intégralement copié dans le module mBoiteDialogue. Les paramètres de l'appel :

7 Quelques états

7.1 Un premier choix

Le formulaire fMenuListes permet de paramétrer les types de contacts concernés par l'état qui résultera du clic sur l'un des boutons.



À l'origine toutes les cases sont cochées.

Un clic sur le bouton « Tous » recoche celles que l'utilisateur aurait décochées.

```
1 Private Sub BtTous_Click()
2     Dim ctl As Control
3     For Each ctl In Me.Controls
```

```
OuvrirUnFichier(Me.hwnd, "Quel document ?", 1, , CurrentProject.Path & "\COURRIER")
Le titre de la boîte
On récupère le chemin complet
On commence la recherche dans le sous-répertoire « Courrier ».
(Dans l'exemple, l'application était logée dans le répertoire « TUTORIELASSOCIATION ».)
```

21-40 : lors d'un clic sur le bouton OK, on vérifie d'abord que l'utilisateur a complété les champs utiles pour publier. 25 : la fonction IsNull(Me.LeNomDuContrôle) renverra « 0 » ou « -1 » selon que le champ a été complété ou non. Donc IsNull(Me.txtCheminDoc) + IsNull(Me.txtMessMail) + IsNull(Me.txtObjet) différent de zéro signifie qu'au moins un champ a la valeur Null.

37-39 : pour chacune des colonnes « Par poste » et « Par e-mail », pour autant qu'il y reste une case cochée, on appelle les procédures PubliPapier ou PubliMail déjà décrites plus haut.

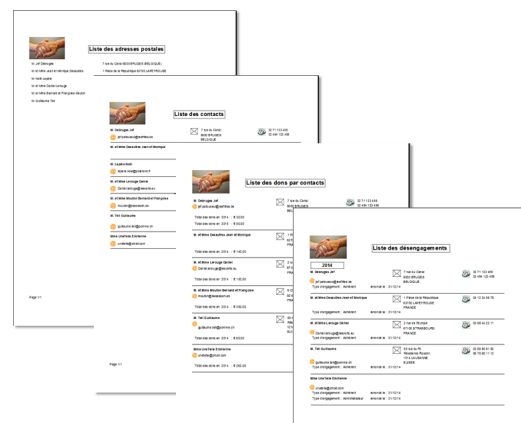
42-51 : lors d'un clic sur le bouton « Tous », on parcourt tous les contrôles du formulaire et s'il s'agit d'un des filtres (leur nom commence par « cc »), on coche la case et on redéclenche le processus prévu à l'ouverture du formulaire.

53-71 : lorsque l'état d'un des filtres est modifié, on redéclenche le processus prévu à l'ouverture du formulaire.

73-fin : avant d'accepter que l'utilisateur coche une case dans les colonnes « Par poste » et « Par email », on vérifie que le contact concerné dispose bien d'une telle adresse. Si ce n'est pas le cas, on affiche un message pour signaler le refus de l'action.

```
4 If ctl.Name Like "cc*" Then
5     ctl = -1
6 End If
7 Next ctl
8 End Sub
```

7.2 Le code associé aux listes présentées ici



Peu de commentaire.

À l'ouverture, on aménage les adresses des images liées (même code que pour un formulaire)

```
1 Private Sub Report_Open(Cancel As Integer)
2     Call AmnImages
3 End Sub
```

Dans l'événement « Sur formatage » de la section qui contient les adresses et téléphone, ce code

```
1 Private Sub Détail_Format(Cancel As Integer, FormatCount As Integer)
2     Me.ImageCourriel.Visible = Not IsNull(Me.txtCourriel1)
3     Me.ImageAdresse.Visible = Not IsNull(Me.txtAdresse1)
4     Me.ImageTel.Visible = (IsNull(Me.txtTelfixe1) + IsNull(Me.txtGsm1)) > -2
5 End Sub
```

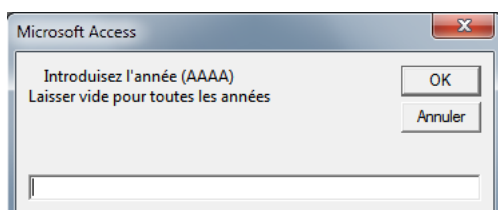
pour cacher l'icône si elle s'avère inutile.

Lorsqu'une année intervient dans l'état (par exemple dans « Liste des dons par contact »), le code est un peu plus étoffé :

- on invite l'utilisateur à saisir une année

```
1 'Choix de l'Année
2 Annee = InputBox("    Introduisez l'année (AAAA)" & vbLf & "Laisser vide pour toutes les années")
```

cette fenêtre apparaît



- on construit alors à la volée la requête source de l'état, par exemple

```
1 'Aménager la source
2 sSelect = "SELECT Format([DonDate], "
3 & "[CiviliteCourte] & " " " & [
4 & "tContacts.Adresse1, tContacts.Adresse
5 & "tContacts.Courriel1, tContacts.
6 & "tContacts.ContactPrenom " _
7 & " FROM tPays RIGHT JOIN ((tCivilites
8 & " ON tContacts.tContactsPK =
9 & "ON tCivilites.tCivilitesPK =
10 & "ON tContacts.tContactsPK = tEngts.
    tContactsFK) ON tPays.tPaysPK =
    tContacts.tPaysFK " _
```

```
11 & "WHERE (((tEngts.tTypesEngtFK) = IIf([
12 & "Or (tEngts.tTypesEngtFK) = IIf([
13 & "Or (tEngts.tTypesEngtFK) = IIf([
14 & "Or (tEngts.tTypesEngtFK) = IIf([
15 & "Or (tEngts.tTypesEngtFK) = IIf([
16 & "Or (tEngts.tTypesEngtFK) = IIf([
17 & "Or (tEngts.tTypesEngtFK) = IIf([
18 & "GROUP BY Format([DonDate], "yyyy"),
19 & "tContacts.Adresse1, tContacts.Adresse
20 & "tPays.Pays, tContacts.Courriel1,
21 & "tContacts.ContactNom, tContacts.
22
23 'Clause Having (dépend du choix de l'
24 If Len(Annee) = 4 Then 'L'utilisateur
25 sHaving = "HAVING Format([DonDate
26 Else
27 sHaving = "HAVING Format([DonDate
28 End If
29 'Affecter la source
30 Me.RecordSource = sSelect & sHaving &
    ";"
```

Pour visualiser la représentation graphique d'un SQL écrit dans le code, vous pouvez utiliser l'interface graphique QBE (Query by Example).

Procédez comme ceci :

- ajouter cette instruction après la confection du SQL, ici, juste avant la dernière ligne

Debug.print sSelect & sHaving & ";"
- afficher l'état pour que le code se déclenche. Vous pouvez alors copier le SQL généré dans la fenêtre d'exécution et le coller dans une requête de test.

Pour vous familiariser avec cette technique, voyez ce tutoriel Initiation - Débogage : requêtes écrites par VBA (lien 168) de Charles A (cafeine) et singulièrement, le chapitre V.



8 Deux bases : une dorsale et une frontale



Pour approfondir le sujet, voyez le tutorial de Morgan Billy (Dolphy35) Comment utiliser une application en mode multiutilisateur : lien 169.

Les tables permanentes et leurs relations, dans la dorsale **AssociationData.mdb** et tout le reste (requêtes, formulaires, états, macros et code) dans la frontale **AssociationSoft.mdb**.

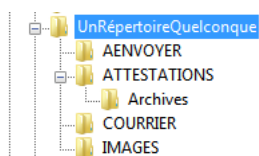
Les deux db sont logées dans le même répertoire. Cette organisation permet de répartir la mise

à jour des tables entre plusieurs utilisateurs. Par exemple l'un se charge de l'enregistrement des dons (fCaisse et fBanque) et l'autre de tout ce qui est administratif (fContacts, envoi des attestations...).

Chacun prend la main tour à tour. Quand l'un a terminé sa gestion, il envoie **AssociationData.mdb** via e-mail à l'autre et lui cède la main. Celui qui prend la main, remplace le fichier AssociationData.mdb qu'il avait sur son PC par celui qu'il vient de recevoir par e-mail. Il opère ses changements et recède la main. Et on est reparti pour un tour...

9 Une arborescence particulière

Ceci pour faciliter la maintenance et alléger le code (utilisation de répertoires relatifs)



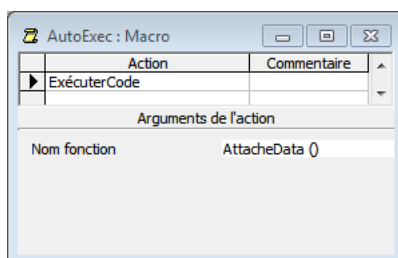
Les deux db, **AssociationSoft.mdb** et **AssociationData.mdb**, sont logées dans un répertoire quelconque (sur le disque dur, sur un réseau, sur une clé USB...).

Ce répertoire contient des sous-répertoires :

AENVOYER : qui recueillera le résultat du dernier publipostage à mettre sous enveloppe; **ATTESTATIONS** : contient le document modèle pour le publipostage des attestations fiscales. Son sous-répertoire Archives contiendra la copie annuelle des attestations destinée au contrôle fiscal éventuel; **COURRIER** : les différents modèles de lettres à publiposter; **IMAGES** : les images liées aux formulaires et états.

9.1 Adaptation automatique de la liaison dorsale/frontale, si changement de localisation

À chaque ouverture de la base, la macro « Autoexec » se déclenche



La fonction **AttacheData()** est logée dans le module **mAttacherTables**.

```

1 Option Compare Database
2 Option Explicit
3
4 Public Function AttacheData()
5 On Error GoTo GestionErreurs
6 Dim db As Database
7 Dim tdfLoop As TableDef
8 Dim CheminSoft As String
9 Dim CheminData As String
10 CheminSoft = CurrentDb.Name
11 CheminData = Left(CheminSoft, Len(
12     CheminSoft) - 8) & "data.mdb"
13 'Vérifie que l'emplacement n'a pas changé
14     depuis la fois précédente
15 If DLookup("Database", "MSysObjects", "
16     not isnull(Database)") = CheminData
17     Then Exit Function 'le répertoire n
18     'a pas été changé
19 'Pour attacher les tables de data
20 Set db = OpenDatabase(CheminData)
21 For Each tdfLoop In db.TableDefs
22     If Left(tdfLoop.Name, 4) <> "
23     MSys" Then
24         DoCmd.DeleteObject acTable,
25             tdfLoop.Name
26         DoCmd.TransferDatabase acLink
27             , "Microsoft Access",
28             CheminData, acTable,
29             tdfLoop.Name, tdfLoop.
30             Name
31     End If
32     Next tdfLoop
33 MsgBox "L'attachement des tables a été
34     actualisé"
35 db.Close
36 Set db = Nothing
37 Exit Function
38 GestionErreurs:
39 If Err.Number = 7874 Then 'la table n'
40     est pas encore dans XXXSoft
41     Resume Next
42 Else
43     MsgBox "Erreur dans l'attachement
44     des tables " & Err.Number & "
45     " & Err.Description
46 End If
47 End Function

```

Explication du code

4 : AttacheData() ne renvoie pas de valeur, ce devrait donc être une **Sub**. On la définit néanmoins

comme *Function* pour pouvoir la déclencher dans une macro (« AutoExec » en l'occurrence).

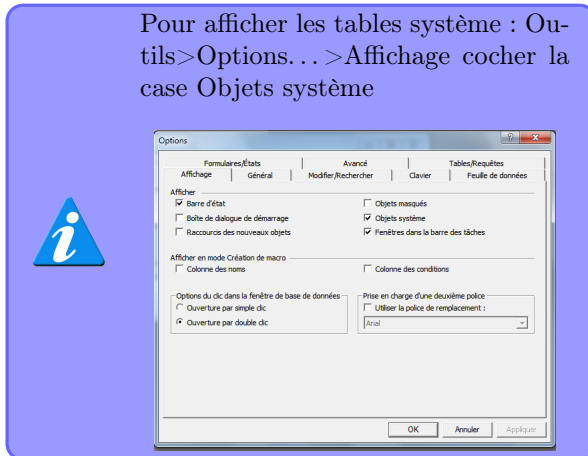
11 : on détermine quel devrait être le chemin de la xxxData.mdb, sachant qu'elle doit se trouver dans le même répertoire que la xxxSoft.mdb.

13 : on vérifie que ce chemin est bien celui enregistré dans la table système *MsysObjects*.

Connect	Database	DateCreate	DateUpdate	Flags	ForeignName	ID
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3102483554
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3147483555
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3147483556
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3147483557
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3147483558
C:\TUTORIELASSOCIATION\AssociationData.mdb	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	2010-11-21 13:51:21	AssociationData	3147483559

Lià où se trouvait la xxxData lors de la précédente fermeture

Pour afficher les tables système : Outils>Options...>Affichage cocher la case Objets système



La table *MsysObjects* contient une colonne « Database » qui renseigne l'adresse de la db qui contient chaque table liée

10 Téléchargement

La db en version Access2000 se trouve ici : lien 170

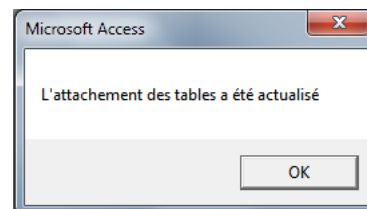
Retrouvez l'article de *Claude Leloup* en ligne : lien 171

Si on constate que l'adresse renseignée correspond à celle « calculée » en 11, on en conclut que les db se trouvent encore au même endroit que lors de l'exécution précédente. Tout va bien, on sort.

Par contre s'il y a différence, c'est que xxx-Soft.mdb a changé d'emplacement et qu'il faut donc réactualiser les liaisons.

15-21 : réactualisation des liaisons. Pour chaque table (autre que les tables système) contenue dans xxxData.mdb, on supprime la table (liée) dans xxx-Soft.mdb (18) et on établit la nouvelle liaison (19).

22 : on signale que le rafraîchissement des liaisons a été effectué :





Excel

Les derniers tutoriels et articles

Les fichiers Excel binaires : xlsb

Alors que depuis la version d'Office 2007, la plupart des utilisateurs et développeurs ne manipulent que des fichiers avec ou sans macros (xlsx ou xlsm), il me semble intéressant de leur proposer un autre choix qui pourrait leur permettre d'améliorer les performances, les tailles et le stockage de données : le format binaire xlsb.

1 Les formats de fichiers Excel

On peut considérer l'évolution récente des fichiers Excel en deux étapes principales : il y en a une avant et une après Excel 2007.

1.1 Avant Excel 2007

1.1.a xls

.xls est l'extension historique des fichiers Microsoft Excel. Ce format a désormais pour nom officiel « Classeur Microsoft Excel 97-2003 ». Ce type de fichier a eu plusieurs caractéristiques de tailles possibles au cours du temps :

- jusqu'à la version 95 (Office 7.0) : 16 384 lignes sur 256 colonnes ($2^{14} \times 2^8$) ;
- jusqu'à la version 2003 (Office 11.0) : 65 536 lignes sur 256 colonnes ($2^{16} \times 2^8$).

De par cette extension, il est impossible a priori de savoir si le fichier Excel comporte ou non du code VBA qui pourrait s'autoexécuter à l'ouverture (procédure `Workbook_Open()`). Cette difficulté à savoir par avance ce que peut contenir un fichier Excel a débouché à partir de la version 2007 sur plusieurs extensions plus spécifiques.

1.2 Depuis Excel 2007

1.2.a xlsx

.xlsx est l'extension des fichiers Excel depuis 2007 sans macro. Il est au format Office Open XML (ou OOXML, norme ISO/CEI 29500). Ce format de fichier ayant pour nom actuel (avril 2015) « Classeur Excel » comporte désormais 1 048 576 lignes par 16 384 colonnes ($2^{20} \times 2^{14}$, soit 17 179 869 184 cellules - plus de 17 milliards de cellules -).

1.2.b xlsm

.xlsm est l'extension des fichiers Excel depuis 2007 avec macros. Également au format Open XML,

il présente les mêmes caractéristiques que le fichier d'extension .xlsx. Les macros sont directement incorporées dans le fichier.

1.3 Autres formats de fichiers

1.3.a xlt,xltx,xltn

Les extensions .xlt, .xltx et .xltn sont dédiées aux fichiers modèles d'Excel. De la même façon que pour les fichiers .xls, les extensions .xltx et .xltn servent respectivement aux fichiers modèles depuis 2007 sans macro et avec macros.

1.3.b xla,xlam

Les extensions .xla et .xlam concernent les fichiers de macros complémentaires (a pour add-in). L'extension .xlax n'existe pas, car de par la nature même de ces fichiers, ils contiennent obligatoirement des macros. La version .xlam existe depuis la version Excel 2007.

1.3.c xlb

L'extension .xlb est utilisée pour les fichiers de configuration, notamment pour les barres d'outils et barres de commandes. Ces paramètres sont modifiables et permettent de savoir quelles barres sont visibles, leurs positions et leurs fonctions.

1.3.d xll

L'extension .xll est utilisée pour les bibliothèques dll spécifiques à Excel. Elle est utilisée lors du développement dans d'autres langages que le VBA (notamment le C++ ou le C#, voire le VB.Net). Un fichier ayant cette extension est dit compilé, c'est-à-dire qu'il n'est pas possible d'accéder directement au code qui s'exécute.

1.3.e xlw

L'extension .xlw permet d'ouvrir un ou plusieurs classeurs d'un seul coup. Le w vient du mot anglais *Workspace* - espace de travail. Le fichier xlw

ne contient pas les données des classeurs, il contient uniquement l'enveloppe des classeurs. De plus, vous devez avoir accès à l'intégralité des classeurs contenus dans le fichier xlw pour pouvoir l'ouvrir correctement.

2 Spécificités techniques du format xlsb

Les fichiers au format xlsb ont les mêmes capacités que les autres formats xlsx et xlsx. Attention toutefois de bien prendre en compte l'absence de

compatibilité Office Open XML, qui empêchera la lecture dans certains cas de figure, et ne permet plus d'utiliser notamment les rubans dynamiques.

3 Comparatif du format xlsb vs. xls/xlsx/xlsm

3.1 Taille des fichiers

En partant d'un fichier d'extension .xls qui contient des données, des formules, des graphiques

et potentiellement des macros, voici les différentes tailles de fichiers constatées après l'enregistrement dans les autres formats :

Taille en Mo	Format xls	Format xlsx	Format xlsm	Format xlsb
Données, formules, graphiques, sans macro	262	88.2	88.2	17.4
Données, formules, graphiques, avec macros	212	Impossible	55.8	38

On peut donc constater que la taille du fichier diminue fortement en utilisant le format xlsb. Vous aurez également remarqué qu'un classeur sans macro d'extension xlsx ou xlsm a exactement la même taille. Les différences de taille sont expliquées par le fait que les informations ne sont pas exactement stockées sous la même forme, voir les liens dans la webographie.

et potentiellement des macros, ainsi que les graphes et formules. Toutes ont la possibilité d'avoir un ruban personnalisé. Néanmoins, de par le caractère non XML du fichier .xlsb, celui-ci ne permet pas de modifier le ruban. La seule solution serait de basculer le fichier .xlsb en .xlsm, de mettre à jour le ruban et de repasser en .xlsb. Enfin, l'incompatibilité XML du format xlsb peut poser des problèmes dans le cadre d'interactions avec les serveurs web par exemple. À noter également que les macros sont parfaitement supportées sous xlsb et qu'enfin, une formule de plus de 8 192 caractères peut être enregistrée correctement avec ce format.

3.2 Limites des contenus

Les trois extensions les plus récentes (.xlsx, .xlsm et .xlsb) ont la même capacité de stockage des don-

4 Conclusions

On a donc pu mettre en avant, dans le cadre de ces tests, le gain de taille du fichier, sa capacité identique pour stocker les informations, les graphiques ou les macros lorsqu'il y en a. Pour les utilisateurs qui manipulent des fichiers de grande taille,

dont la compatibilité XML n'est pas requise, et pour lesquels aucun ruban personnalisé n'est à compter, le format xlsb est la solution la plus performante à adopter !

5 Webographie

Quelques liens expliquant notamment les différences de poids de fichiers : New Binary File Format

for Spreadsheets ([lien 172](#)) et Office 2007 .bin file format ([lien 173](#))

Retrouvez l'article de **Jean-Philippe André** en ligne : [lien 174](#)

Liste des liens

Page 2

- lien 1 : ... http://www.boost.org/doc/libs/1_49_0/libs/mpl/doc/index.html
- lien 2 : ... <http://www.artima.com/cppsource/metafunctions.html>
- lien 3 : ... <http://2012.cppnow.org/session/metaprogramming-11/>

Page 4

- lien 4 : ... <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2003/n1489.pdf>

Page 5

- lien 5 : ... <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3351.pdf>
- lien 6 : ... <http://code.google.com/p/origin/>
- lien 7 : ... <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2004/n1720.html>
- lien 8 : ... <http://akrzemi1.developpez.com/metaprogrammation-metafonctions-cpp-11/>

Page 6

- lien 9 : ... <http://www.codeproject.com/Articles/396959/Mario>

Page 15

- lien 10 : ... <https://github.com/FlorianRappl/Mario5TS>
- lien 11 : ... <http://www.codeproject.com/Articles/396959/Mario>
- lien 12 : ... <http://www.codeproject.com/Articles/432832/Editor-for-Mario>
- lien 13 : ... <http://yahiko.developpez.com/tutoriels/TypeScript-Mario5/demo.zip>
- lien 14 : ... <http://yahiko.developpez.com/tutoriels/TypeScript-Mario5/source.zip>
- lien 15 : ... <https://github.com/FlorianRappl/Mario5TS>
- lien 16 : ... <http://yahiko.developpez.com/tutoriels/TypeScript-Mario5/>

Page 16

- lien 17 : ... <https://angularjs.org/>

Page 19

- lien 18 : ... <http://marcautran.developpez.com/tutoriels/angular/angularmvc/>

Page 20

- lien 19 : ... <http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/>
- lien 20 : ... <https://developers.google.com/closure/compiler/>
- lien 21 : ... <http://www.scala-js.org/doc/>
- lien 22 : ... <https://groups.google.com/forum/?fromgroups#!forum/scala-js>
- lien 23 : ... <https://gitter.im/scala-js/scala-js>
- lien 24 : ... http://fr.wikipedia.org/wiki/Scala_%28langage%29
- lien 25 : ... <http://www.scala-lang.org/old/node/168>
- lien 26 : ... <http://www.developpez.net/forums/d1508527/java/general-java/langage/scala/scala-js-nouveau-compileur-javascript-scala/>

Page 24

- lien 27 : ... <http://docs.oracle.com/javafx/>
- lien 28 : ... <http://java.developpez.com/faq/javafx/>
- lien 29 : ... <http://fxexperience.com/>
- lien 30 : ... <http://mikarber.developpez.com/tutoriels/java/introduction-javafx/>
- lien 31 : ... <http://www.jetbrains.com/>
- lien 32 : ... <http://www.jetbrains.com/idea/>
- lien 33 : ... <http://www.eclipse.org/>
- lien 34 : ... <http://www.netbeans.org/>
- lien 35 : ... http://www.jetbrains.com/idea/features/editions_comparison_matrix.html
- lien 36 : ... <http://www.jetbrains.com/idea/download/>

Page 25

- lien 37 : ... <http://gruntjs.com/>
- lien 38 : ... <http://www.eclipse.org/mylyn/>
- lien 39 : ... http://www.jetbrains.com/idea/features/tasks_and_context.html

lien 40 : ... <http://linsolas.developpez.com/articles/java/outils/intellij-idea-13/#LIII-D>
lien 41 : ... <http://www.jetbrains.com/idea/features/index.html>
lien 42 : ... http://jadclipse.sourceforge.net/wiki/index.php/Main_Page

Page 26

lien 43 : ... <https://www.jetbrains.com/idea/help/@contract-annotations.html>

Page 27

lien 44 : ... <http://linsolas.developpez.com/articles/java/outils/intellij-idea-13/#LIII>
lien 45 : ... http://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard.pdf
lien 46 : ... http://www.jetbrains.com/idea/docs/IntelliJIDEA_ReferenceCard_Mac.pdf
lien 47 : ... http://www.ptxstore.com/jetbrains/product_info.php?products_id=1638
lien 48 : ... <http://plugins.jetbrains.com/plugin/6546>
lien 49 : ... http://www.jetbrains.com/idea/features/eclipse_java.html
lien 50 : ... <http://java.developpez.com/faq/intellijidea>
lien 51 : ... <http://www.developpez.net/forums/f1871/java/edi-outils-java/autres-edi/intellij/>
lien 52 : ... <http://confluence.jetbrains.com/display/IntelliJIDEA/IntelliJ+IDEA+for+Eclipse+Users>
lien 53 : ... <http://confluence.jetbrains.com/display/IntelliJIDEA/IntelliJ+IDEA+for+NetBeans+Users>

Page 28

lien 54 : ... <https://www.jetbrains.com/webstorm/>
lien 55 : ... <http://www.jetbrains.com/idea/>
lien 56 : ... <http://www.jetbrains.com/idea/features/index.html>
lien 57 : ... <http://www.jetbrains.com/idea/whatsnew/index.html>
lien 58 : ... <http://java.developpez.com/faq/intellijidea>
lien 59 : ... <http://linsolas.developpez.com/articles/java/outils/intellij-idea-13/>
lien 60 : ... <http://www.developpez.net/forums/f1871/java/edi-outils-java/autres-edi/intellij/>
lien 61 : ... <http://linsolas.developpez.com/articles/java/outils/intellij-idea-14/>

Page 29

lien 62 : ... <http://alain-bernard.developpez.com/tutoriels/eclipse/decouverte-emf-edit/fichiers/src-emf-edit.zip>
lien 63 : ... <http://alain-bernard.developpez.com/tutoriels/eclipse/decouverte-emf-edit/fichiers/src-emf-edit.zip>
lien 64 : ... <http://mbaron.developpez.com/eclipse/introemf/>
lien 65 : ... <http://www.vogella.com/tutorials/EclipseEMF/article.html>

Page 41

lien 66 : ... <http://alain-bernard.developpez.com/tutoriels/eclipse/databinding-swt/>

Page 45

lien 67 : ... <http://alain-bernard.developpez.com/tutoriels/eclipse/sirius-intro>
lien 68 : ... http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.emf.doc%2Fpreferences%2Foverview%2FEMF.Edit.html&cp=21_0_1
lien 69 : ... <http://fr.slideshare.net/mikaelbarbero/emfedit>
lien 70 : ... <http://eclipse-source.com/blogs/tutorials/getting-started-with-EMF-Forms/>
lien 71 : ... <http://wiki.eclipse.org/EEF>
lien 72 : ... <http://projects.eclipse.org/projects/modeling.emf-parsley>
lien 73 : ... <http://alain-bernard.developpez.com/tutoriels/eclipse/decouverte-emf-edit/>

Page 46

lien 74 : ... <http://www.developpez.net/forums/d1504864/java/general-java/java-mobiles/android/android-5-1-disponible/>

Page 47

lien 75 : ... <http://www.developpez.com/actu/80113/Android-Lollipop-peine-a-convaincre-Kitkat-continue-son-ascension/>
lien 76 : ... <http://www.developpez.net/forums/d1492776/java/general-java/java-mobiles/android/google-ne-publiera-plus-patchs-securite-webview-versions-anterieures-android-4-4-kitkat/>

Page 48

lien 77 : ... <http://olegoaer.developpez.com/cours/mobile/>

Page 50

lien 78 : ... <http://nicroman.developpez.com/tutoriels/android/exceptions/>

lien 79 : ... <http://www.developpez.net/template/images/logo.png>

lien 80 : ... <http://chart.apis.google.com/chart?cht=qr&chs=200x200&chl=Coucou>

Page 51

lien 81 : ... <http://loripsum.net/api/plaintext/short/20/>

Page 52

lien 82 : ... <http://my.company.com/remote/logUserConnexion?id=00000000-54b3-e7c7-0000-000046bfd97&event=CONNECT>

lien 83 : ... <http://olegoaer.developpez.com/tutos/mobile/android/rpc/fichiers/RPC.zip>

lien 84 : ... <http://olegoaer.developpez.com/tutos/mobile/android/rpc/>

Page 54

lien 85 : ... <http://hiko-seijuro.developpez.com/articles/ddd/>

lien 86 : ... <http://loulou.developpez.com/tutoriels/cpp/debogueur-visual-studio/>

Page 61

lien 87 : ... <http://man.developpez.com/man3/gets/>

lien 88 : ... <http://c.developpez.com/faq/?page=Gestion-du-clavier-et-de-l-ecran-en-mode-console#Comment-vider-le-buffer-clavier>

lien 89 : ... <http://www.developpez.net/forums/d1470116/c-cpp/c/debuter/j-ai-probleme-menu-s-affiche-double/>

Page 63

lien 90 : ... <http://man.developpez.com/man3/assert/>

Page 64

lien 91 : ... <http://valgrind.org/>

lien 92 : ... <http://www.drmemory.org/>

Page 66

lien 93 : ... <http://alexandre-laurent.developpez.com/articles/regles-or-programmation/>

Page 67

lien 94 : ... <http://alexandre-laurent.developpez.com/articles/debogage-application/>

Page 68

lien 95 : ... <http://opengl.developpez.com/docs/man/man/glbegint>

lien 96 : ... <http://opengl.developpez.com/docs/man/man2/glend>

lien 97 : ... <http://khronos.org/>

lien 98 : ... <http://jeux.developpez.com/>

lien 99 : ... <http://jeux.developpez.com/index/rss>

lien 100 : ... <http://twitter.com/2D3DJeuxDVP>

lien 101 : ... <http://www.facebook.com/pages/2D-3D-Jeux/109575245738013>

lien 102 : ... <http://jeux.developpez.com/actu/71696/Un-developpeur-donne-son-avis-sur-la-conception-d-OpenGL-et-explique-pourquoi-OpenGL-est-en-retard-par-rapport-a-DirectX-12-ou-Mantle/>

Page 69

lien 103 : ... <http://jeux.developpez.com/actu/64149/Mantle-une-nouvelle-bibliotheque-graphiques-pour-mieux-controler-le-GPU-visant-a-se-debarrasser-des-obstacles-complicant-la-vie-des-developpeurs/>

lien 104 : ... <http://jeux.developpez.com/actu/69137/DirectX-12-a-ete-presente-a-la-GDC-2014-la-bibliotheque-arrivera-durant-l-annee-2015/>

lien 105 : ... <http://jeux.developpez.com/actu/71870/WWDC-Apple-devoile-Metal-une-nouvelle-bibliotheque-graphique-pour-ameliorer-les-performances-de-rendu-CPU-sur-les-peripheriques-iOS/>

lien 106 : ... <http://jeux.developpez.com/actu/74343/Decouvrez-quelques-detaills-supplementaires-sur-la-prochaine-version-d-OpenGL-et-comment-la-bibliotheque-comblera-ses-lacunes/>

lien 107 : ... <http://jeux.developpez.com/actu/74179/La-prochaine-version-d-OpenGL-pourrait-integrer-Mantle-et-ainsi-etre-aussi-performante-que-Direct3D-12/>

lien 108 : ... <http://jeux.developpez.com/actu/82056/Le-successeur-d-OpenGL-s-appelle-Vulkan-decouvrez-ce-que-sera-la-nouvelle-bibliotheque-de-hautes-performances-pour-les-GPU/>

lien 109 : ... <http://opengl.developpez.com/docs/man/man/glUniform>
lien 110 : ... <https://www.khronos.org/registry/spir-v/specs/1.0/SPIRV.pdf>

Page 70

lien 111 : ... Plusieurs sociétés (Valve, LunarG, Codeplay) travaillent déjà sur des outils. Voici GLAVE, un outil de débogage développé par Valve et LunarG :
lien 112 : ... Imagination Technologies a porté une de leurs démonstrations OpenGL ES 3.0 vers Vulkan :
lien 113 : ... Valve, grâce au pilote Linux open source d'Intel, a pu montrer une démonstration du moteur Source 2 utilisant
lien 114 : ... <http://vulkan.developpez.com/videos/GDC/2015/>
lien 115 : ... <https://www.khronos.org/vulkan>
lien 116 : ... http://vulkan.developpez.com/fichiers/GDC-2105/Khronos-Vulkan-GDC_Mar15.pdf
lien 117 : ... http://vulkan.developpez.com/fichiers/GDC-2105/Valve-Vulkan-Session-GDC_Mar15.pdf
lien 118 : ... <https://www.khronos.org/assets/uploads/developers/library/2015-gdc/Khronos-Vulkan-GDC-Mar15.pdf>
lien 119 : ... <https://www.khronos.org/registry/spir-v/papers/WhitePaper.pdf>
lien 120 : ... <http://vulkan.developpez.com/articles/presentation/>

Page 71

lien 121 : ... <http://doc-snapshots.qt.io/qt5-5.5/qt3d-index.html>
lien 122 : ... <http://qt.developpez.com/index/redirect/23586/Apercu-de-Qt-3D-2-0-presentation-generale-du-futur-moteur-3D-de-Qt-un-article-de-Sean-Harmer/>
lien 123 : ... <http://doc-snapshots.qt.io/qt5-5.5/qtcanvas3d-index.html>
lien 124 : ... <http://doc-snapshots.qt.io/qt5-5.5/qtlocation-index.html>

Page 72

lien 125 : ... <https://wiki.qt.io/New-Features-in-Qt-5.5>
lien 126 : ... <https://wiki.qt.io/Qt-5.5.0-tools-and-versions>
lien 127 : ... http://download.qt.io/development_releases/qt/5.5/5.5.0-alpha/
lien 128 : ... <http://bugreports.qt.io/>
lien 129 : ... <http://www.developpez.net/forums/d1507015/c-cpp/bibliotheques/qt/sortie-qt-5-5-beta/>

Page 73

lien 130 : ... <https://www.jetbrains.com/clion/>
lien 131 : ... <http://www.developpez.net/forums/d1515563/c-cpp/bibliotheques/qt/edi/qt-creator/sortie-qt-creator-3-4-0-a/>
lien 132 : ... <https://fr.wikipedia.org/wiki/Shader>
lien 133 : ... https://fr.wikipedia.org/wiki/Occlusion_ambiante
lien 134 : ... https://fr.wikipedia.org/wiki/High_dynamic_range_rendering
lien 135 : ... https://fr.wikipedia.org/wiki/Deferred_Shading
lien 136 : ... <http://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342>
lien 137 : ... https://en.wikipedia.org/wiki/Texture_mapping
lien 138 : ... https://www.opengl.org/wiki/Vertex_Rendering#Instancing
lien 139 : ... https://www.opengl.org/wiki/Uniform_Buffer_Object

Page 74

lien 140 : ... https://fr.wikipedia.org/wiki/Moteur_physique
lien 141 : ... https://fr.wikipedia.org/wiki/Détection_de_collision
lien 142 : ... https://en.wikipedia.org/wiki/3D_audio_effect
lien 143 : ... <https://fr.wikipedia.org/wiki/Rigging>
lien 144 : ... https://en.wikipedia.org/wiki/Morph_target_animation
lien 145 : ... https://fr.wikipedia.org/wiki/Recherche_de_chemin
lien 146 : ... https://fr.wikipedia.org/wiki/Intelligence_artificielle#Jeux_vid.C3.A9o
lien 147 : ... https://fr.wikipedia.org/wiki/Space_Invaders

Page 76

lien 148 : ... <http://www.kdab.com/>
lien 149 : ... <http://www.kdab.com/overview-qt3d-2-0-part-1/>

lien 150 : ... <http://kdab.developpez.com/tutoriels/qt-3d-2/01-presentation-generale/>

Page 77

lien 151 : ... <http://python.developpez.com/cours/>

lien 152 : ... <http://ceg.developpez.com/tutoriels/pyqt/qt-quick-python/01-interfaces-simples/>

lien 153 : ... <http://ceg.developpez.com/tutoriels/pyqt/qt-quick-python/01-interfaces-simples/>

lien 154 : ... <http://ceg.developpez.com/tutoriels/python/configurer-qtcreator-pour-python/>

lien 155 : ... <http://qt-devnet.developpez.com/tutoriels/qt-quick/pour-developpeurs-cpp/>

lien 156 : ... <http://doc.qt.io/qt-5/qtquickcontrols-index.html>

Page 78

lien 157 : ... <http://doc.qt.io/qt-5/qmltypes.html>

Page 81

lien 158 : ... <http://ceg.developpez.com/tutoriels/pyqt/qt-quick-python/02-interaction-qml-python/>

Page 82

lien 159 : ... <http://mhubiche.developpez.com/Access/cours/bases/>

lien 160 : ... <http://jeannot45.developpez.com/articles/access/creationrequetes1/>

lien 161 : ... <http://argyronet.developpez.com/office/access/highlightrecord/#L2-1>

lien 162 : ... <http://heureuxoli.developpez.com/office/word/vba-all/>

Page 83

lien 163 : ... <http://claudeleloup.developpez.com/tutoriels/access/comment-implanter-un-sous-formulaire/>

Page 84

lien 164 : ... <http://claudeleloup.developpez.com/tutoriels/access/comment-garder-images-hors-base-donnees-access/>

Page 85

lien 165 : ... <http://claudeleloup.developpez.com/tutoriels/access/outil-comptable/#LIV>

Page 86

lien 166 : ... <http://claudeleloup.developpez.com/tutoriels/access/emettre-archiver-documents-types/#LV-A-1>

Page 95

lien 167 : ... <http://access.developpez.com/faq/?page=CheminsRep#AffBoitDialog>

Page 96

lien 168 : ... <http://cafeine.developpez.com/access/tutoriel/debugprint/>

Page 97

lien 169 : ... <http://dolphy35.developpez.com/article/access/basesreseau/#LII>

Page 98

lien 170 : ... <http://claudeleloup.developpez.com/tutoriels/access/association/AssociationPGM.zip>

lien 171 : ... <http://claudeleloup.developpez.com/tutoriels/access/association/>

Page 100

lien 172 : ... <http://blogs.msdn.com/b/dmahugh/archive/2006/08/22/712835.aspx>

lien 173 : ... <http://www.arstdesign.com/articles/office2007bin.html>

lien 174 : ... <http://jpcheck.developpez.com/tutoriels/excel/fichiers-excel-binaires-xlsb/>