



Developpez

Le Mag

Édition de août - septembre 2012.

Numéro 41.

Magazine en ligne gratuit.

Diffusion de copies conformes à l'original autorisée.

Réalisation : Alexandre Pottiez

Rédaction : la rédaction de Developpez

Contact : magazine@redaction-developpez.com

Sommaire

JavaScript	Page 2
DotNet	Page 6
C++	Page 9
Qt	Page 13
Access	Page 40
Java	Page 49
Eclipse	Page 52
NetBeans	Page 53
Android	Page 54
Liens	Page 56

Article C++



Retour de fonctions ou exceptions ?

Souvent, vous avez hésité entre l'utilisation d'une simple valeur de retour ou d'une exception. Pire, vous trouvez que la mise en place d'un code prévenant toutes erreurs est lourde. Cet article est là pour vous aider à apprendre comment écrire un code simple et sain.

par **Alexandre Laurent**

Page 9



Article Access

Éditorial

Voici votre tout nouveau manuel à imprimer et à emporter partout.

Vous pourrez réviser toutes vos technologies préférées avant vos cours.

Profitez-en bien !

La rédaction

Comment rechercher une valeur dans une table qui contient des paliers ?

Il s'agit de rechercher dans une table, une valeur qui ne s'y trouve pas nécessairement et de choisir selon les circonstances.

par **Claude Leloup**

Page 40

L'interactivité avec la balise HTML5 Canvas

L'élément HTML5 <canvas> donne aux développeurs des possibilités d'interactivité et de dessin jusqu'alors inaccessibles sans l'utilisation de programmes tiers tels que Flash. Dans cet article, nous allons considérer le problème de l'interactivité et comment l'utiliser avec les balises HTML5 <canvas>. Vous pourrez utiliser ces informations pour créer un jeu HTML5 simple.

1. Jouer

Jusqu'il y a peu, il était difficile d'imaginer (et de créer) des jeux codés uniquement en HTML et JavaScript. Avec la balise <canvas>, la notion de jeu ou simplement de zone interactive devient envisageable, sans avoir besoin de Flash, qui était traditionnellement considéré comme la seule solution pratique. Pour commencer, attardons-nous sur la gestion du clavier dans un canvas.

2. Les entrées clavier

La balise <canvas> se comporte comme toute autre balise. Elle possède des attributs width, height et id. Le contenu de la balise correspond à un contenu alternatif pour les navigateurs ne la supportant pas. J'ai aussi défini une fonction à exécuter au chargement de la page et une balise <div> qui contiendra certaines informations, à savoir la quantité de mouvements effectués par l'élément. **Nous allons créer un carré que l'on pourra déplacer avec les touches fléchées du clavier.** Bien sûr, vous pourrez vous inspirer de cet exemple pour créer beaucoup d'autres fonctionnalités.

```
<body onload="init();" >

  <div>
    Vous avez bougé de <span id="x"
onload="counting(true);">0</span> horizontalement
et de <span id="y" onload="counting();">0</span>
verticalement !
    <br />
    Ne laissez pas la boîte gagner ! (Utiliser
les touches fléchées)
  </div>

  <canvas id="game" width="640" height="500">
    Oh non ! Votre navigateur ne reconnaît pas la
balise <canvas>.
  </canvas>

</body>
```

2.1. Le JavaScript

Le code JavaScript est un petit peu plus compliqué. En JavaScript, on peut savoir quelle touche a été pressée en utilisant e.keyCode. Chaque caractère du clavier possède un identifiant prédéfini que vous pouvez retrouver. Nous souhaitons créer un carré qui se déplace, nous aurons donc besoin des touches fléchées. Pour commencer, copiez le code suivant dans votre fichier JavaScript. Il s'agit juste des variables dont nous aurons besoin.

```
var canvas, draw, width, height; // Pour dessiner
dans le canvas
var downcheck, upcheck, rightcheck, leftcheck; //
Pour déterminer la direction

// Position courante
var up = 0;
var down = 0;
var left = 0;
var right = 0;
```

Ensuite, nous devons créer une fonction init(), qui détermine ce qu'il faut faire au chargement de la page.

```
// La fonction principale pour initialiser le
tout
function init() {

  // Récupérer le canvas à partir de son id
  canvas = document.getElementById('game');

  // Définir les dimensions de l'objet créé
dans le canvas
  height = canvas.height;
  width = canvas.width;

  // Une variable utilisée pour dessiner
l'objet actuel
  draw = canvas.getContext('2d');

  // Nous voulons actualiser l'affichage toutes
les 30 ms
  setInterval(redraw, 30);

  // Lorsqu'une touche est appuyée, lancer une
fonction ;
  // en lancer une autre lorsque la touche est
relâchée
  document.onkeydown = canvasMove;
  document.onkeyup = canvasStop;
}
```

Déplacer des rectangles dans un canvas peut être compliqué (le canvas étant dessiné autant de fois que demandé mais pas supprimé, vous vous retrouverez rapidement avec tout un tas de rectangles superposés), nous allons le redessiner toutes les 30 millisecondes. Cette fonction est appelée dans init().

```
// Effacer le rectangle quand nous devons le
déplacer afin de pouvoir le redessiner.
function clear(c) {
  c.clearRect(0, 0, width, height);
}
```

```

}

function redraw() {
    clear(draw);
    draw.fillStyle = 'rgba(0,0,0,0.5)';
    draw.fillRect(left - right , down - up,
'100', '100');
}

```

Ensuite, nous devons ajouter une fonctionnalité pour déplacer le carré. Nous voulons que deux touches fléchées puissent être utilisées ensemble, nous devons donc prévoir un fonctionnement un peu plus complexe qu'une simple instruction `if()`. Nous ajustons les variables de test des touches à `true` ou `false` selon les touches appuyées par l'utilisateur, de cette façon, nous pouvons avoir deux variables fixées à `true` en même temps, ce qui autorise les déplacements en diagonale.

```

function canvasMove(e) {

    // Vérifie si la touche haut est appuyée
    if(e.keyCode == '38') upcheck = true
    // Sinon, vérifier si la touche gauche est
    appuyée et si haut est true
    // Dans ce cas, mettre la flèche gauche à
    true
    else if(e.keyCode == '37' && upcheck == true)
leftcheck = true;
    // else check if the right arrow is true,
    repeat.
    else if(e.keyCode == '39' && upcheck == true)
rightcheck = true;
    // otherwise the up arrow is not being
    pressed, so it is false
    else upcheck = false;

    // Repeat for every arrow direction
    if(e.keyCode == '40') downcheck = true;
    else if(e.keyCode == '37' && downcheck ==
true) leftcheck = true;
    else if(e.keyCode == '39' && downcheck ==
true) rightcheck = true;
    else downcheck = false;

    if(e.keyCode == '37') leftcheck = true;
    else if(e.keyCode == '40' && leftcheck ==
true) downcheck = true;
    else if(e.keyCode == '38' && leftcheck ==
true) upcheck = true;
    else leftcheck = false;

    if(e.keyCode == '39') rightcheck = true;
    else if(e.keyCode == '40' && rightcheck ==
true) downcheck = true;
    else if(e.keyCode == '38' && rightcheck ==
true) upcheck = true;
    else rightcheck = false;

    // If the variables are true, increase the
    left and right movement accordingly.
    // We also run the counting function, to keep
    track of total movement.
    if(rightcheck == true) { left += 10;
counting() };
    if(leftcheck == true) { right += 10;
counting() };
    if(upcheck == true) { up += 10;
counting(true) };
    if(downcheck == true) { down += 10;
counting(true) };
}

```

```

}

```

Ensuite, nous devons vérifier si l'utilisateur relâche une touche et affecter les variables correspondantes à `false`.

```

// Lorsque l'utilisateur relâche une touche,
mettre la variable correspondante à false
function canvasStop(e) {

    if(e.keyCode == '38') {
        upcheck = false;
    }
    if(e.keyCode == '40') {
        downcheck = false;
    }
    if(e.keyCode == '37') {
        leftcheck = false;
    }
    if(e.keyCode == '39') {
        rightcheck = false;
    }
}
}

```

Pour finir, nous avons une fonction pour comptabiliser le tout.

```

function counting(y) {

    if(y == true) {
        document.getElementById('y').innerHTML =
parseInt(document.getElementById('y').innerHTML)
+ 1;
    }

    else {
        document.getElementById('x').innerHTML =
parseInt(document.getElementById('x').innerHTML)
+ 1;
    }
}
}

```

Et voilà ! Si vous souhaitez voir tout cela en action, faites un tour sur la page de démo ([Lien 01](#)) ou téléchargez l'exemple complet ([Lien 02](#)).

3. Les entrées de souris

Maintenant, nous allons créer une interface de dessin et cela est étonnamment facile avec `canvas`. Comme d'habitude, nous allons commencer avec un peu de HTML.

```

<body onload="init();">
<div id="drawingcontainer">
    <div class="text">
        Dessinez ci-dessous !
    </div>
    <div id="options">
        <input type="submit" value="1px"
onclick="changeSize(1);" />
        <input type="submit" value="2px"
onclick="changeSize(2);" />
        <input type="submit" value="3px"
onclick="changeSize(3);" />
        <input type="submit" value="5px"
onclick="changeSize(5);" />
        <input type="submit" value="10px"
onclick="changeSize(10);" />
        <input type="submit" value="15px"
onclick="changeSize(20);" />
    </div>
</div>

```

```

        <input type="submit"
onclick="changeColor('#871de0');" class="purple"
value="" />
        <input type="submit"
onclick="changeColor('#eb159d');" class="pink"
value="" />
        <input type="submit"
onclick="changeColor('#c92626');" class="red"
value="" />
        <input type="submit"
onclick="changeColor('#db551b');" class="orange"
value="" />
        <input type="submit"
onclick="changeColor('#ddc41e');" class="yellow"
value="" />
        <input type="submit"
onclick="changeColor('#76b80f');" class="green"
value="" />
        <input type="submit"
onclick="changeColor('#1974b4');" class="blue"
value="" />
        <input type="submit"
onclick="changeColor('#000');" class="black"
value="" />
        <input type="submit"
onclick="changeColor('#FFF');" class="white"
value="" />
    </div>

    <canvas width="690" height="500"
id="drawing">
    Your browser doesn't support canvas
    </canvas>
</div>
</body>

```

Ensuite, définissons quelques variables et lançons la fonction `init()`.

```

var canvas, draw;

var started = false;
var x, y;

function init() {

    // Récupérer la zone de dessin
    canvas = document.getElementById('drawing');
    context = canvas.getContext('2d');

    // Ajoutons des gestionnaires d'événements
    pour savoir ce qu'il se passe
    // et lançons quelques fonctions associées.
    canvas.addEventListener('mousemove',
    mousemovement, false);
    canvas.addEventListener('mousedown',
    mouseclick, false);
    canvas.addEventListener('mouseup',
    mouseunclick, false);
}

```

Maintenant, nous avons besoin de quelques fonctions pour effectuer des actions lorsque l'utilisateur clique, bouge la souris puis relâche la souris. Tout ce que font ces fonctions est de connaître la position de la souris et de tracer une ligne lorsque c'est utile.

```
function mouseclick() {
```

```

    // Lorsque le clic est détecté, passe la
    variable started à true
    // et déplace la position initiale de la
    souris
    context.beginPath();
    context.moveTo(x, y);
    started = true;
}
// For getting the mouse position, basically.
This gets the position
// of the canvas element, so we can use it to
calculate the mouse
// position on the canvas
// Récupération de l'emplacement de la souris.
// On récupère la position de l'élément canvas
pour pouvoir
// récupérer la position de la souris à
l'intérieur du canvas.
function getOffset(e) {
    var cx = 0;
    var cy = 0;

    while(e && !isNaN(e.offsetLeft) && !
    isNaN(e.offsetTop)) {
        cx += e.offsetLeft - e.scrollLeft;
        cy += e.offsetTop - e.scrollTop;
        e = e.offsetParent;
    }
    return { top: cy, left: cx };
}

```

```

function mousemovement(e) {

    // Récupérer la position de la souris

    if(e.offsetX || e.offsetY) {
        x = e.pageX -
        getOffset(document.getElementById('drawing')).lef
        t - window.pageXOffset;
        y = e.pageY -
        getOffset(document.getElementById('drawing')).top
        - window.pageYOffset;
    }
    else if(e.layerX || e.layerY) {
        x = (e.clientX + document.body.scrollLeft
        + document.documentElement.scrollLeft)
        -
        getOffset(document.getElementById('drawing')).lef
        t - window.pageXOffset;
        y = (e.clientY + document.body.scrollTop
        + document.documentElement.scrollTop)
        -
        getOffset(document.getElementById('drawing')).top
        ;
    }

    // Si la variable started vaut true, alors
    tracer une ligne
    if(started) {
        contextMouse.lineTo(x, y);
        contextMouse.stroke();
    }
}

```

```

function mouseunclick() {
    // Passer la variable started à false lorsque
    le bouton est relâché
    if(started) {
        started = false;
    }
}

```

```
}  
}
```

Enfin, nous avons besoin de deux fonctions supplémentaires pour modifier la largeur et la couleur du trait.

```
// Changer la taille  
function changeSize(s) {  
    context.lineWidth = s;  
}  
  
// Changer la couleur  
function changeColor(c) {  
    context.strokeStyle = c;  
}
```

Et voilà ! Pour finir, ajoutons un peu de style en CSS.

```
body {  
    margin: 0;  
    padding: 0;  
    background: url('background.gif');  
    font-family: Arial, Helvetica, sans-serif;  
}  
  
canvas {  
    box-shadow: 0px 0px 30px rgba(0,0,0,0.3);  
    float: left;  
    border-radius: 6px;  
    display: block;  
}  
  
div {  
    font-family: 'Droid Sans', Helvetica, Arial,  
    sans-serif;  
    font-size: 30px;  
    padding: 15px;  
    line-height: 30px;  
}  
  
#options {  
    float: left;  
}  
  
h3 {  
    float: left;  
    margin: 0;  
}  
  
div span {  
    float: left;  
    text-transform: uppercase;  
    padding: 0 20px 20px 20px;  
    font-size: 14px;  
    font-weight: bold;  
}
```

```
div input {  
    float: left;  
    display: inline;  
}  
  
div input[type=range] {  
    position: relative;  
    top: 6px;  
}  
  
#options input[type=submit] {  
    border: 0;  
    border-radius: 4px;  
    margin-right: 10px;  
    width: 35px;  
    box-shadow: inset 0px 0px 10px  
    rgba(0,0,0,0.2);  
    height: 35px;  
    position: relative;  
    cursor: pointer;  
}  
  
#options input[type=submit]:hover {  
    box-shadow: inset 0px 0px 10px  
    rgba(0,0,0,0.5);  
}  
  
#options input[type=submit]:active {  
    top: 2px;  
}  
  
.purple { background: #871de0; }  
.pink { background: #eb159d; }  
.red { background: #c92626; }  
.orange { background: #db551b; }  
.yellow { background: #ddc41e; }  
.green { background: #76b80f; }  
.blue { background: #1974b4; }  
.black { background: #000; }  
.white { background: #fff; }  
  
#drawingcontainer {  
    width: 710px;  
    margin: 0px auto;  
}
```

J'ai mis en place une page de démonstration ([Lien 01](#)) pour tester tout cela. Regardez la page de démonstration ([Lien 01](#)) ou téléchargez l'archive ([Lien 02](#)) si vous souhaitez modifier le script.

Cet article a été traduit avec l'aimable autorisation de inserthtml. L'article original : A Look at HTML5 Canvas Interactivity peut être vu sur le site de inserthtml : [Lien 03](#).

Retrouvez l'article de Johnny Simpson traduit par Didier Mouronval en ligne : [Lien 04](#)

Strings en C# et .NET

Le type System.String (raccourci string en C#) est l'un des types les plus importants en .NET, et malheureusement, il est très mal compris. Cet article tente d'expliquer certaines bases de ce type.

1. Traduction

Ceci est la traduction la plus fidèle possible de l'article de Jon Skeet, Strings in C# and .NET : [Lien 05](#).

2. Qu'est-ce qu'un string ?

Un string est essentiellement une séquence de caractères. Chaque caractère est un caractère Unicode ([Lien 06](#)) dans la plage U+0000 à U+FFFF (nous aborderons ce sujet plus tard). Le type string (j'utiliserai la forme raccourcie de C# plutôt que System.String à chaque fois) possède les caractéristiques décrites aux paragraphes suivants.

C'est un type référence

C'est une idée fautive très répandue que le string est un type valeur. C'est parce que son immuabilité (voir le point suivant) fonctionne un peu comme un type valeur. Il agit en fait comme un type référence normal. Vous pouvez voir mes articles sur le passage de paramètre ([Lien 07](#)) et la mémoire ([Lien 08](#)) pour plus de détails sur les différences entre les types valeur et référence.

Il est immuable

Vous ne pouvez jamais réellement changer le contenu d'un string, du moins avec du code sécurisé qui n'utilise pas la réflexion. Pour cette raison, vous finissez souvent par modifier la valeur d'une *variable* de type string. Par exemple, le code `s = s.Replace("foo", "bar");` ne change pas le contenu du string auquel `s` faisait référence à l'origine - cela affecte la valeur de `s` à une nouvelle chaîne de caractères qui est une copie de l'ancien string, mais avec "foo" remplacé par "bar".

Il peut contenir des valeurs NULL

Les programmeurs C sont habitués à ce que les strings soient des séquences de caractères se terminant par '\0', le caractère nul ou null. (Je vais utiliser "null" car c'est ce que le tableau de code Unicode indique en détail, ne le confondez avec le mot-clé null en C# - char est un type valeur et ne peut donc pas être une référence null !). En .NET, les chaînes peuvent contenir des caractères null sans aucun problème en ce qui concerne les méthodes de string. Cependant, d'autres classes (par exemple un grand nombre de classes Windows Forms) peuvent très bien penser que le string se termine au premier caractère null - si votre chaîne apparaît curieusement tronquée, cela pourrait être le problème.

Il surcharge l'opérateur ==

Lorsque l'opérateur `==` est utilisé pour comparer deux chaînes de caractères, la méthode `Equals` est appelée, ce qui vérifie l'égalité des contenus des chaînes de caractères plutôt que les références en elles-mêmes. Par exemple, `"hello".Substring(0, 4)=="hell"` est vrai, même si les références des deux côtés de l'opérateur sont différentes (elles se réfèrent à deux objets strings différents, qui contiennent tous les deux la même séquence de caractères). Notez que la surcharge d'opérateur ne fonctionne ici que si les deux côtés de l'opérateur sont des strings lors de la compilation - les opérateurs ne sont pas appliqués de manière polymorphe. Si chaque côté de l'opérateur est de type objet, dans la mesure où le compilateur est concerné, l'opérateur de base `==` sera appliqué et la simple égalité de référence sera testée.

3. Pool interne

Le framework .NET a le concept du "pool interne". Il s'agit juste d'un ensemble de strings, mais il s'assure qu'à chaque fois que vous faites référence à la même chaîne littérale, vous obtenez une référence à la même chaîne. Ceci est probablement dépendant du langage, mais c'est certainement vrai en C# et VB.NET, et je serais très surpris de voir un langage pour lequel ce ne serait pas le cas, l'IL rend cela très facile à faire (probablement plus facile que de ne pas réussir à interner des littéraux). Les littéraux sont automatiquement internés, mais on peut également les interner manuellement via la méthode `Intern`, et vérifier si oui ou non il y a déjà une chaîne internée avec la même séquence de caractères dans le pool en utilisant la méthode `IsInterned`. Cette méthode renvoie une chaîne de caractères plutôt qu'un boolean - si une chaîne de caractères équivalente est dans le pool, une référence à cette chaîne est renvoyée. Sinon, la valeur null est renvoyée. De même, la méthode `Intern` renvoie une référence à un string interné - soit le string que vous avez passé s'il était déjà dans le pool, ou un nouveau string interné créé, ou un string équivalent qui était déjà dans le pool.

4. Littéraux

Les littéraux correspondent à la façon dont vous codez vos strings dans vos programmes écrits en C#. Il existe deux types de littéraux en C# : les littéraux de chaîne normaux et les littéraux de chaîne verbatim. Les littéraux de chaîne normaux sont semblables à ceux que l'on retrouve dans d'autres langages tels que Java et C - ils commencent et se terminent par " et divers caractères (en particulier, " lui-même, \, le retour chariot (CR) et le saut de ligne (LF)) ont besoin d'être "échappés" afin d'être représentés dans la chaîne de caractères. Les littéraux de chaîne verbatim permettent à peu près tout en leur sein et se terminent au

premier " qui n'est pas doublé. Même les retours chariot et les sauts de ligne peuvent apparaître dans le littéral ! Pour obtenir un " au sein de la même chaîne, vous aurez besoin d'écrire "". Les littéraux de chaîne verbatim se distinguent par la présence d'un @ devant les guillemets d'ouverture. Voici quelques exemples des deux types de littéraux :

Littéral de chaîne normal	Littéral de chaîne verbatim	Résultat
"Hello"	@"Hello"	Hello
"Backslash: \\"	@"Backslash: \"	Backslash: \
"Quote: \\""	@"Quote: \"\""	Quote: "
"CRLF:\r\nPost CRLF"	@"CRLF: Post CRLF"	CRLF: Post CRLF

Notez que la différence n'a de sens que pour le compilateur. Une fois que le string est dans le code compilé, il n'y a pas de différence entre les littéraux de chaîne normaux et les littéraux de chaîne verbatim.

L'ensemble des séquences d'échappement est :

- \ ' - simple quote, requis pour les caractères littéraux ;
- \" - double-quote, requis pour les strings littéraux ;
- \\ - antislash ;
- \0 - caractère 0 Unicode ;
- \a - alerte (caractère 7) ;
- \b - backspace (caractère 8) ;
- \f - saut de page (caractère 12) ;
- \n - nouvelle ligne (caractère 10) ;
- \r - chariot retour (caractère 13) ;
- \t - tabulation horizontale (caractère 9) ;
- \v - quote verticale (caractère 11) ;
- \uxxxx - séquence d'échappement Unicode pour les caractères ayant une valeur hexa xxxx ;
- \xn[n][n][n] - séquence d'échappement Unicode pour les caractères ayant une valeur hexa nnnn (la longueur de la variable est une version de \uxxxx) ;
- \Uxxxxxxx - séquence d'échappement Unicode pour les caractères ayant une valeur hexa xxxxxxxx (pour la génération de substituts).

Parmi ceux-ci, \a, \f, \v, \x et \U sont rarement utilisés, d'après ma propre expérience.

5. Les strings et le débogueur

De nombreuses personnes rencontrent des problèmes en inspectant des strings dans le débogueur, avec VS.NET 2002 et VS.NET 2003. Ironiquement, les problèmes sont souvent générés par le débogueur qui essaie d'être utile et soit affiche le string comme un littéral de chaîne normale échappé d'un anti-slash, soit l'affiche en tant que littéral de chaîne verbatim précédé d'un @. Cela conduit à de nombreuses questions demandant comment le @ peut être enlevé, en dépit du fait que ce n'est pas vraiment là sa première place - c'est seulement la façon dont le débogueur le montre. En outre, certaines versions de VS.NET arrêteront d'afficher le contenu de la chaîne de caractères au premier caractère null, évalueront sa propriété Length de manière incorrecte et calculeront la

valeur elles-mêmes au lieu de faire appel au code managé. Encore une fois, ces versions considèrent que le string s'arrête au premier caractère null.

Compte tenu de la confusion que cela a causée, je crois qu'il est préférable d'examiner les strings d'une façon différente lorsqu'on est en mode debug, du moins si vous avez l'impression que quelque chose de bizarre se produit. Je vous suggère d'utiliser une méthode comme celle indiquée ci-dessous, qui va afficher le contenu du string dans la console d'une manière sûre. Selon le type d'application que vous développez, vous voudrez peut-être écrire cette information dans un fichier de log, dans le debug ou trace de listener, ou l'afficher dans une message box.

```
static readonly string[] LowNames =
{
    "NUL", "SOH", "STX", "ETX", "EOT", "ENQ", "ACK",
    "BEL",
    "BS", "HT", "LF", "VT", "FF", "CR", "SO", "SI",
    "DLE", "DC1", "DC2", "DC3", "DC4", "NAK", "SYN",
    "ETB",
    "CAN", "EM", "SUB", "ESC", "FS", "GS", "RS",
    "US"
};

public static void DisplayString (string text)
{
    Console.WriteLine ("String length: {0}",
        text.Length);
    foreach (char c in text)
    {
        if (c < 32)
        {
            Console.WriteLine ("<{0}> U+{1:x4}",
                LowNames[c], (int)c);
        }
        else if (c > 127)
        {
            Console.WriteLine ("(Possibly non-printable)
                U+{0:x4}", (int)c);
        }
        else
        {
            Console.WriteLine ("{0} U+{1:x4}", c, (int)c);
        }
    }
}
```

6. Usage de la mémoire

Dans l'implémentation actuelle, les strings prennent 20+ (n/2)*4 bytes (arrondi à la valeur n/2 près), où n est le nombre de caractères du string. Le type string est inhabituel en ceci que la taille de l'objet varie elle-même. Les seules autres classes qui permettent cela (pour autant que je sache) sont les tableaux. Essentiellement, un string est un tableau de caractères en mémoire, plus la longueur du tableau et la longueur du string (en caractères). La longueur du tableau n'est pas toujours la même que celle des caractères, vu que les strings peuvent être "suralloués" au sein de la bibliothèque mscorlib.dll, afin de les construire plus facilement. (StringBuilder peut le faire, par exemple.) Alors que les strings sont immuables, pour le monde extérieur, le code au sein de mscorlib peut changer les contenus, de sorte que StringBuilder crée un string avec un tableau de caractères interne plus grand que le contenu actuel l'exige, puis ajoute des caractères à ce

string jusqu'à ce que la taille du tableau soit atteinte, à ce stade il crée un nouveau string avec un grand tableau. La propriété Length de string contient également un flag dans son premier bit pour dire s'il contient ou pas des caractères non ASCII. Cela permet une optimisation supplémentaire dans certains cas.

Bien que les strings ne se terminent pas par null dans la mesure où l'API est concernée, le tableau de caractères se termine par null, ce qui lui permet d'être passé directement à des fonctions non managées sans qu'une copie ne soit impliquée, en supposant que l'inter-op indique que le string peut être marshallé en tant qu'Unicode.

7. Encodage

Si vous ne savez pas ce qu'est l'encodage des caractères Unicode, vous pouvez lire mon article dédié à ce sujet : [Lien 09](#).

Comme indiqué au début de l'article, les strings sont toujours encodés en Unicode. L'idée "d'une chaîne Big-5" ou d'une "chaîne encodée en UTF-8" est une erreur (dans la mesure où .NET est concerné) et indique généralement un manque de compréhension, soit de l'encodage, soit de la manière dont .NET gère les strings. Il est très important de comprendre cela - le traitement d'un string comme s'il représentait un texte valide dans un encodage non Unicode est presque toujours une erreur.

Maintenant, le jeu de caractères Unicode (un des défauts de l'Unicode est que ce terme est utilisé pour différentes choses, incluant un jeu de caractères codés et un système de codage de caractères) contient plus de 65 536 caractères. Cela signifie qu'un seul char (System.Char) ne peut pas couvrir tous les caractères. Cela conduit à l'utilisation de substituts où les caractères ci-dessus U+FFFF sont représentés dans les strings par deux caractères. Essentiellement, un string utilise le système d'encodage de caractères UTF-16. La plupart des développeurs n'ont pas vraiment besoin d'en savoir plus à ce sujet, mais cela vaut au moins le coup d'en être conscient.

8. Les bizarreries de la culture et de l'internationalisation

Quelques-unes des bizarreries de l'Unicode conduisent à des bizarreries dans la gestion des strings et des caractères. Bon nombre des méthodes de string sont dépendantes de la *culture* - en d'autres termes, ce qu'ils font dépend de la culture du thread courant. Par exemple, vous vous attendez à ce que "i".ToUpper() renvoie quoi ? La plupart des gens diraient "I", mais en turc, la bonne réponse serait "İ" (Unicode U+130, "Lettre latine I majuscule avec un point dessus"). Pour effectuer un changement de casse indépendamment de la culture, vous pouvez utiliser CultureInfo.InvariantCulture et le passer à la surcharge de

String.ToUpper qui prend un CultureInfo.

Il y a d'autres anomalies quand il s'agit de comparer, trier et de trouver l'index d'une sous-chaîne. Certaines d'entre elles sont spécifiques à la culture et d'autres pas. Par exemple, dans toutes les cultures (d'après ce que j'ai pu constater) "lassen" et "la\u00dfen" (un "scharfes S" ou eszett étant le caractère Unicode échappé ici) sont considérés comme étant égaux lorsque CompareTo ou Compare sont utilisés, mais pas lorsque c'est Equals. IndexOf traitera le eszett de la même manière qu'un "ss", sauf si vous utilisez un CultureInfo.IndexOf et que vous spécifiez CompareOptions.Ordinal comme option à utiliser.

Certains autres caractères Unicode semblent être invisibles pour le IndexOf normal. Quelqu'un a demandé sur le newsgroup C# pourquoi une méthode recherche/remplace partirait dans une boucle infinie. Il utilisait Replace de façon répétée pour remplacer tous les doubles espaces par un seul espace, et vérifiait si c'était terminé ou non en utilisant IndexOf, de sorte que les espaces multiples étaient réduits à un espace. Malheureusement, cela échouait à cause d'un "étrange" caractère dans le string original entre deux espaces. IndexOf trouvait les doubles espaces, ignorant le caractère supplémentaire, mais pas Replace. Je ne sais pas quel était le vrai caractère présent dans le string, mais le problème peut facilement être reproduit en utilisant U+200C qui est un caractère antiliant sans chasse (quoi que cela puisse vouloir dire : exactement !). Mettez un de ces caractères au milieu du texte que vous recherchez et IndexOf va l'ignorer, mais pas Replace. Une fois encore, pour que les deux méthodes se comportent de la même façon, vous pouvez utiliser CultureInfo.IndexOf et lui passer un CompareOptions.Ordinal. Je suppose qu'il y a beaucoup de codes qui échoueraient face à des données "bizarres" comme celles-ci (je n'aurai pas la prétention de dire que tout mon code est à l'abri de cela).

Microsoft a quelques recommandations concernant la gestion des strings - elles datent de 2005, mais sont toujours valables : [Lien 10](#).

9. Conclusion

Pour un type aussi fondamental, les strings (et les données textuelles en général) sont plus complexes que ce à quoi on pourrait s'attendre. Il est important de comprendre les bases évoquées ici, même si quelques-uns des points les plus fins de comparaison dans des contextes multiculturels vous échappent pour le moment. En particulier, être en mesure de diagnostiquer les erreurs de code où les données sont perdues, en vérifiant les *vraies* valeurs des strings, est essentiel.

Retrouvez l'article de Jon Skeet traduit par Jean-Michel Ormes en ligne : [Lien 11](#)

Retour de fonctions ou exceptions ?

Souvent, vous avez hésité entre l'utilisation d'une simple valeur de retour ou d'une exception. Pire, vous trouvez que la mise en place d'un code prévenant toutes erreurs est lourde. Cet article est là pour vous aider à apprendre comment écrire un code simple et sain.

1. Article original

Cet article est la traduction du billet Return-code vs. Exception handling par Aaron Lahman : [Lien 12](#).

2. Introduction

C'est une des guerres de religion dans la théorie des langages de programmation : faut-il choisir de retourner un code d'erreur (cette méthode sera appelée par la suite RCH, pour Return Code Handling) ou utiliser une exception (appelée par la suite EH, pour Exception Handling) ? Premièrement, mon point de vue est biaisé. Deuxièmement, j'essaierai de rester le plus objectif possible, cloisonnant mes observations sur du code utilisant ces mécanismes en fonction de ce que je pense de leur utilité. En général, je préfère utiliser les exceptions et je suis l'adage disant que « les exceptions sont à limiter aux choses exceptionnelles ».

Cela étant dit...

Raymond Chen a écrit, il y a quelque temps, un article sur le sujet : [Lien 13](#). Il y indique avec exactitude quelques faits sur la revue de code suivant l'une ou l'autre approche, concluant qu'il est généralement plus difficile de déterminer la fiabilité d'un code utilisant les retours de fonctions, que celle d'un code utilisant les exceptions pour propager les erreurs (Raymond Chen conclut en fait toute autre chose : qu'il est plus simple de reconnaître un code non robuste quand il suit l'approche des RCH. En effet, un tel code ne présente pas un *if* toutes les deux lignes. Il est donc indubitablement mauvais).

Il a raison. Et il a tort. Son article est incomplet. Commençons par définir le terrain de jeu.

3. Retours de fonction

Ci-dessous, nous avons deux versions d'un code utilisant la valeur de retour de fonction. Toutes les erreurs sont communiquées grâce à un code de retour. La première version est un exemple erroné, les chemins d'erreurs ne sont pas pris en compte. La deuxième version corrige cette faute.

```

BOOL ComputeChecksum(LPCTSTR pszFile, DWORD*
pdwResult)
{
    HANDLE h = INVALID_HANDLE;
    HANDLE hfm = INVALID_HANDLE;
    void *pv;
    DWORD dwHeaderSum;

```

```

    h = CreateFile(...);

    hfm = CreateFileMapping(h, ...);

    pv = MapViewOfFile(hfm, ...);

    CheckSumMappedFile(... &dwHeaderSum,
pdwResult);

    UnmapViewOfFile(pv);
    CloseHandle(hfm);
    CloseHandle(h);
    return TRUE;
}

```

```

BOOL ComputeChecksum(LPCTSTR pszFile, DWORD*
pdwResult)
{
    HANDLE h = INVALID_HANDLE;
    HANDLE hfm = INVALID_HANDLE;
    void *pv = NULL;
    DWORD dwHeaderSum;
    BOOL result = FALSE;

    if ( (h = CreateFile(...)) == INVALID_HANDLE)
        goto cleanup;
    if ( (hfm = CreateFileMapping(h, ...)
        == INVALID_HANDLE ) )
        goto cleanup;
    if ( (pv = MapViewOfFile(hfm, ...) == NULL )
        goto cleanup;

    CheckSumMappedFile(... &dwHeaderSum,
pdwResult);
    result = TRUE;

cleanup:
    if (pv) UnmapViewOfFile(pv);
    if (hfm != INVALID_HANDLE) CloseHandle(hfm);
    if (h != INVALID_HANDLE) CloseHandle(h);
    return result;
}

```

Le premier code semble très propre mais ne vérifie pas les retours de fonction. Donc, selon la méthode de Chen, il est indubitablement incorrect. Toujours selon Chen, le premier code est « très facile » à écrire et il est « très facile » de remarquer qu'il est incorrect. L'absence de **if toutes les deux lignes** est un signal d'alarme. En gros, vous devez tester tous les chemins (on appelle un chemin, chaque cas d'exécution possible du code) que ce soit de réussite ou d'échec, même les cas non intéressants.

Passons au code d'exemple utilisant les exceptions du billet de Chen.

4. Exceptions (ancienne approche)

Ci-dessous se trouve une adaptation du code de son article en utilisant les exceptions. Le deuxième code est une correction possible du deuxième code.

```
NotifyIcon* CreateNotifyIcon()
{
    NotifyIcon* icon;

    icon = new NotifyIcon();

    icon.set_text("Blah blah blah");
    icon.set_icon(new Icon(...), GetInfo());
    icon.set_visible(true);

    return icon;
}
```

```
NotifyIcon* CreateNotifyIcon()
{
    NotifyIcon* icon = 0;

    icon = new NotifyIcon();
    try {
        icon.set_text("Blah blah blah");
        icon.set_visible(true);
        Info info = GetInfo();
        icon.set_icon(new Icon(...), info);
    } catch (...) {
        delete icon; throw;
    }
    return icon;
}
```

Je suis en accord avec n'importe quelle critique disant que le code ci-dessus est difficile à diagnostiquer. Si l'une des fonctions `set_text`, `set_icon` ou `set_visible` peut lancer une exception, le premier code est totalement fichu. Le second code est délicat... en déplaçant `GetInfo()` sur une nouvelle ligne et `set_icon` à la fin, l'objet `Icon` ne peut pas provoquer de fuites (en supposant que `set_icon` soit responsable de la libération de `Icon` en cas d'erreur). Le `try-catch` supprimera `NotifyIcon` et propagera l'exception, mais tout cela rend le programme pénible à déboguer : nous allons voir deux exceptions au lieu d'une seule.

Encore pire, demain, ce code pourrait devenir incorrect. Si le constructeur de copie de `Info` envoie une exception, le second code est toujours faux. Je parie que vous n'y aviez pas pensé. Vous croyez être un expert en cela ? Essayez le point numéro 18 du livre *Exceptional C++* ([Lien 14](#)). Je pense que la première fois, j'ai trouvé 16 parcours d'exécution possibles dans trois lignes de code, me faisant entrer dans la catégorie 'Guru', mais cela m'a pris quinze minutes (l'exercice 18 consiste à trouver le maximum de chemins d'exécution dans un code de trois lignes. Selon votre résultat, vous appartenez aux catégories suivantes : « Average » pour moins de trois, « Exception-aware » pour un score de quatre à quatorze et « Guru material » pour un score de quinze à vingt-trois). Normalement, vous n'avez que cinq secondes dans une revue de code et vous ne trouverez donc pas les vingt-trois possibilités

d'exécution. Vous avez mieux à faire que d'analyser toutes les possibilités.

Si on applique les critères de Raymond Chen, avoir un code correct est 'très dur' et j'espère que mon auditoire est d'accord avec moi. C'est compliqué. N'importe quelle ligne de code, n'importe quelle sous-expression peut lancer une exception et nous devons surveiller toutes les possibilités. La méthode utilisant les retours de fonction semble meilleure. Bien sûr, nous aurons quelque chose comme neuf *if* pour l'exemple ci-dessus, mais au moins, on verrait que toutes les possibilités d'exécution sont couvertes. Cependant, nous ne pouvons appliquer aucune règle pour voir si le code est trivialement incorrect. Dans la méthode utilisant les retours de fonction, une ligne sans branchement conditionnel est suspecte. Pour la méthode utilisant les exceptions (ancienne méthode), une ligne sans `try-catch` après une acquisition de ressources est suspecte.

J'appelle cela « approche ancienne » car cela ressemble beaucoup à la façon de procéder en C# ou en C++ des années 1990, sans le support de bibliothèques. Stephan T. Lavavej distingue le C++ 'archaïque' et le C++ 'moderne' et Raymond Chen note à la fin : « Oui, il y a des modèles de programmation comme le RAII et les transactions, mais vous voyez rarement un code d'exemple les utilisant. ». Corrigons cela.

5. Exceptions (approche moderne)

En premier, nous avons un mauvais code utilisant les exceptions et le RAII pour reprendre l'exemple de Chen. En second, la version corrigée :

```
shared_ptr<NotifyIcon>
CreateNotifyIcon()
{
    NotifyIcon*
        icon( new NotifyIcon() );

    icon->set_text("Blah blah blah");

    icon->set_icon(
        shared_ptr<Icon>( new Icon(...) ),
        GetInfo());

    icon->set_visible(true);

    return icon;
}
```

```
shared_ptr<NotifyIcon>
CreateNotifyIcon()
{
    shared_ptr<NotifyIcon>
        icon( new NotifyIcon() );

    icon->set_text("Blah blah blah");

    shared_ptr<Icon>
        inner( new Icon(...) );
    icon->set_icon(inner, GetInfo());

    icon->set_visible(true);

    return icon;
}
```

Je change les règles. Je ne me soucie plus du nombre de branchements, que ce soit lors d'échecs ou de réussites. Par contre, je **me préoccupe énormément** du fait qu'à chaque instant, il n'y a qu'un seul propriétaire pour chaque ressource et qu'en cas d'échec, ce propriétaire gèrera la ressource.

C'est un changement fondamental. L'approche moderne **utilisant le RAII n'a pas toujours un code explicite pour les conditions d'échecs**. Les échecs courants sont toujours gérés non localement. La ressource locale est **immédiatement** prise en charge par des objets automatiques, grâce au RAII (« automatique » réfère ici au type de stockage de l'objet (automatique, statique, dynamique et thread local en C++11, FDIS : §3.7.3)), c'est-à-dire qui seront détruits à la sortie du bloc. N'importe quelle ressource brute (comme l'allocation d'un nouvel objet) doit toujours être déléguée à un objet RAII.

Détaillons ce que j'ai trouvé de mauvais dans le premier exemple :

- à la première ligne, nous avons un pointeur nu, mauvais. Règle du RAII : pointeur nu = code suspect ;
- à la deuxième ligne, `set_text`. Aucune ressource brute, bien ;
- à la troisième ligne, `shared_ptr` est mélangé avec un autre appel, mauvais. À déplacer sur une autre ligne ;
- à la dernière ligne, `set_visible`, également correct.

Comment savons-nous que nous avons fini ? Lorsqu'il n'y a que deux sortes de lignes : les initialisations d'objets RAII et le code utile qui ne crée pas de ressources non encapsulées. Maintenant, il est vrai qu'il est dur d'utiliser correctement des classes RAII dès le début, mais cela doit être fait seulement une fois par classe au lieu d'une fois par utilisation.

Bien qu'il soit compliqué de voir toutes les possibilités d'exécution, il devient vraiment facile de vérifier si le développeur a terminé avec l'utilisation des exceptions. N'importe quelle allocation de ressource sans RAII dira que le développeur n'a pas terminé. Bien sûr, un mauvais code utilisant les retours de fonction apparaît comme une rose dans un jardin de mauvaises herbes. Mais avec le RAII, vous pouvez normalement éviter d'écrire du mauvais code dès le départ : passez directement aux `shared_ptr<T>` ou `shared_factory<HANDLE, close_handle>::type` ou `score_guard<T>`, etc. Vous pouvez ainsi éviter d'écrire du mauvais code et gérer toutes les erreurs avec le RAII (Certains remarqueront qu'`auto_ptr` (ou `unique_ptr` en C++0x) peut s'avérer plus efficace et s'autoconvertit sans risques en `shared_ptr`. C'est une alternative possible. N'utilisez juste pas de pointeurs nus et assurez-vous de relire le chapitre sur les `auto_ptr` du livre Effective C++ ([Lien 15](#))).

6. Pensées finales

Nous avons donc un nouveau tableau pour reconnaître le code :

Différences entre bon et mauvais code utilisant RCH	- Y a-t-il un if pour chaque appel de fonction qui peut échouer ? - Y a-t-il un bloc de nettoyage et des goto ? - Est-ce que chaque ressource est libérée dans le bloc de nettoyage ?
Différences entre bon et mauvais code utilisant EH (ancienne approche)	- Quelles lignes peuvent lancer une exception ? Le nombre de throws = le nombre de if dans RCH. - Y a-t-il un bloc catch-throw pour chaque ressource nécessitant d'être libérée ? - Assurez-vous qu'il n'y ait pas de 'goto' : ils cassent les destructeurs C++.
Différences entre bon et mauvais code utilisant EH (approche méthode)	- Est-ce que les objets natifs sont directement stockés dans des objets RAII ? - Est-ce que l'initialisation RAII est effectuée sur une ligne distincte ? - Suspectez toute utilisation de delete, ou l'utilisation d'un new pour les tableaux (préférez un vector à la place).

Enfin, voici mon avis sur comment une utilisation moderne des exceptions et du RAII rentre dans le tableau de Raymond Chen :

Domaine	Facile	Difficile	Très difficile
Faire du mauvais code avec RCH ?	X		
Différenciation bon code mauvais code ?	X		
Faire du bon code avec RCH ?		X	
Faire du mauvais code avec les EH (ancienne approche)			X
Différenciation bon code mauvais code ?			X
Faire du bon code avec EH (ancienne approche) ?			X
Faire du mauvais code avec EH (approche moderne)		X	
Différenciation bon code mauvais code ?	X		
Faire du bon code avec EH (approche moderne) ?	X		

Finalement, voici mon code moderne utilisant les retours de fonction ou les exceptions.

```

HRESULT
CreateNotifyIcon(NotifyIcon** ppResult)
{
    NotifyIcon*    icon = 0;
    Icon*          inner = 0;
    const wchar_t * tmp1 = 0;
    HRESULT        hr = S_OK;

    if ( SUCCEEDED(hr) ) {
        icon = new (nothrow) NotifyIcon();
        if ( !icon ) hr = E_OUTOFMEM;
    }

    if ( SUCCEEDED(hr) )
        hr = icon->set_text("Blah blah blah");

    if ( SUCCEEDED(hr) ) {
        inner = new (nothrow) Icon(...);
        if ( !inner )
            hr = E_OUTOFMEM;
        else {
            Info info;
            hr = GetInfo( &info );

            if ( SUCCEEDED(hr) )
                hr = icon->set_icon(inner, info);
            if ( SUCCEEDED(hr) )
                inner = NULL;
        }
    }
    if ( SUCCEEDED(hr) )
        hr = icon->set_visible(true);
}

```

```

if ( SUCCEEDED(hr) ) {
    *ppResult = icon;
    icon = NULL;
} else {
    *ppResult = NULL;
}

cleanup:
if ( inner ) delete inner;
if ( icon ) delete icon;
return hr;
}

```

```

shared_ptr<NotifyIcon>
CreateNotifyIcon()
{
    shared_ptr<NotifyIcon> icon;
    shared_ptr<Icon>      inner;

    icon.reset( new NotifyIcon() );
    icon->set_text("Blah blah blah");

    inner.reset( new Icon(...) );
    icon->set_icon(inner, GetInfo());
    icon->set_visible(true);

    return icon;
}

```

Retrouvez l'article d'Aaron Lahman traduit par Alexandre Laurent en ligne : [Lien 16](#)



Digia s'investit de plus en plus dans Qt au point de le racheter, Nokia continue-t-il de sombrer ?

Il y a une quinzaine de mois, Nokia « refilait » le support commercial de Qt à Digia ; certaines rumeurs parlaient déjà d'une revente complète de tout ce qui concerne le framework, il ne s'agissait que du support commercial – à l'époque. Ces rumeurs se révèlent maintenant confirmées : Digia s'apprête à acquérir tout ce qui concerne Qt de Nokia (toutes les activités Qt, dont le développement du produit, les licences open source et commerciales, tout le service commercial). Ceci intervient dans une période relativement sombre pour Nokia (« Nokia coule-t-il à pic ? » : [Lien 17](#)), avec parfois des décisions douloureuses et, selon certains, injustifiables (pour se relancer, la société a licencié une grande partie de ses développeurs, selon Mirko Boehm ([Lien 18](#)), se basant probablement sur les annonces d'un « affinage stratégique » ([Lien 19](#))).

Petit historique, assez bref, du framework Qt, généralement annoncé comme le standard *de facto* pour des interfaces graphiques portables en C++ (et d'autres langages) : l'histoire commence en 1992, à Oslo, en Norvège, avec le développement d'un outil pour la gestion d'interfaces graphiques, tant pour Windows que X11, n'étant disponible en open source que pour ce dernier. Quatre ans plus tard, le projet KDE débute : un environnement graphique pour Linux entièrement basé sur Qt. Un accord est signé en 1998, entre Trolltech (éditeur de Qt) et la nouvelle KDE Free Qt Foundation, pour s'assurer que, si Trolltech venait à disparaître ou à ne plus publier de version libre de Qt pendant un an, le framework tombe sous une licence très permissive, une BSD. En 2008, Trolltech est racheté par Nokia, ce dernier voulant l'introduire dans ses plateformes mobiles (Qt est très utilisé pour le développement d'applications pour Symbian et MeeGo) ; en 2011, revirement total de situation : un accord est signé entre Nokia et Microsoft pour relancer les parts de marché du premier sur le monde des smartphones et pour montrer que le second y existe toujours. Peu après, les licences commerciales Qt sont déchargées à Digia ; la même année, le framework commence une migration vers l'*open governance*. Non, Nokia ne veut pas se séparer de Qt, ne revend pas Qt, ne cède pas Qt, juste les licences commerciales : [Lien 20](#). Début de cette année 2012, l'*open governance* est bien là, avec le Qt Project, les révisions de code et autres joyeusetés qui montrent que le projet s'ouvre réellement, qu'une société ne peut pas tout diriger. On s'attendait à voir plus de choses bouger, avec les difficultés de Nokia : en juillet, voyant les habituels Developer Days toujours pas annoncés, la communauté reprend cette partie en charge : [Lien 21](#). En août, Nokia n'a plus grand-chose à voir avec Qt. Cependant, l'accord avec la KDE Free Qt Foundation tient toujours – « au cas où ».

Ces années avec Nokia n'ont pas été les pires du

framework, loin de là : en 2009, le framework est aussi publié sous LGPL ([Lien 22](#)), une licence beaucoup plus permissive (certains diront « plus libre ») que la GPL, ce qui porte à trois le nombre de licences possibles pour l'utilisation du framework. Il faut aussi noter l'apparition de Qt Quick ([Lien 23](#)), révolutionnaire en bien des sens pour le framework : l'accent est mis sur des interfaces plus dynamiques. Le framework s'est répandu sur des millions de périphériques mobiles, comme le 808 PureView (sous Symbian, avec un capteur de... quarante et un mégapixels !) ou le N9 (l'un des rares, si ce n'est le seul smartphone sous MeeGo).

Cependant, avec l'accord avec Microsoft, Qt n'était plus très utile à Nokia, la plateforme Windows Phone étant très orientée vers .Net et le code managé (absolument incompatible avec Qt). Il fallait cependant sécuriser la plateforme (et les revenus ?), tant en tant que projet open source qu'en tant que communauté. C'est ainsi que Digia a été choisi, récupérant par là la technologie Qt, le copyright, les équipes Qt de chez Nokia (au maximum cent vingt-cinq personnes). Déjà très active dans la communauté Qt, Digia ne peut qu'en profiter pour grandir dans l'écosystème Qt, devenant incontournable mais semblant vouloir que tout continue comme avant (ou en mieux) : la distribution tant en open source qu'en commercial leur semble un très bon point, pour couvrir un maximum de besoins des clients, dans toute leur diversité (en ce compris KDE).

Digia a déjà pris les devants dès la passation des licences commerciales effectuée (et même avant), avec notamment le support de systèmes d'exploitation en temps réel bien amélioré (comme QNX avec Qt 4.8.1 ([Lien 24](#)), mais aussi INTEGRITY ou VxWorks) – réalisation du vœu « code once, deploy everywhere », même sans interface graphique. Dès l'acquisition finalisée, Digia prendra les rênes opérationnelles du Qt Project, dont l'hébergement. D'ailleurs, le premier test n'est pas si loin : Qt 5 ne devrait plus trop tarder, une nouvelle version prévue très majeure pour le framework, avec notamment l'arrivée de la modularisation et Qt Quick 2, sans oublier le support des plus conventionnels widgets en C++. Un axe envisagé d'exploration est le support d'autres plateformes mobiles, dont Android et iOS.

Commentez la news de Thibaut Cuvelier en ligne : [Lien 25](#)

La FAQ Qt s'enrichit d'une vingtaine de questions-réponses sur l'utilisation de QxOrm.

Après une bien trop longue absence, la FAQ Qt est de retour, avec une toute nouvelle architecture et une révision de la presque totalité des QR existantes (orthographe, liens, contenu) : ainsi, la FAQ sera bien plus agréable à utiliser ! De plus, une trentaine de nouvelles QR ont été

ajoutées, remplaçant le départ de celles concernant Qt Creator, sans oublier celles concernant QxOrm, un excellent ORM pour Qt, dont Developpez héberge le forum francophone : [Lien 26](#).

Nous essaierons de proposer plus de petites mises à jour dans les mois à venir, *stay tuned* !

Retrouvez la FAQ Qt en ligne : [Lien 27](#)

Les derniers tutoriels et articles

Modéliser en C++, designer avec QML : le jeu de patience

Cet article est un petit tutoriel qui vise à montrer les différents moyens d'interfacer QML avec le C++. On ne rentre pas dans le code C++ qui est supposé connu par le lecteur, on explique les concepts QML et en particulier là où il y a une interaction avec le C++.

1. Introduction

Après avoir lu quelques articles sur le QML, j'étais un peu entre deux eaux : c'est super cool, magique, beau, etc. mais c'est JavaScript, déclaratif, ce n'est plus du vrai C++ avec les classes, les objets. Afin d'en avoir le cœur net, j'ai donc décidé de m'y plonger et d'essayer de faire une application dont l'interface est en QML et dont le modèle, le cœur est en C++. Pour un simple petit jeu comme le solitaire (un jeu de cartes, une réussite classique) QML et JavaScript peuvent certainement suffire, j'ai néanmoins voulu appliquer le principe de séparation de l'interface et du modèle afin d'évaluer QML pour un projet de plus grande envergure. L'arrivée de Qt5 me conforte dans l'idée que c'est une bonne voie, également pour de plus grandes applications bureautiques.

1.1. Un projet réel comme but

Dans ce genre d'apprentissage, il faut un but pas trop compliqué, un objectif que l'on peut garder dans le viseur afin d'intégrer les concepts et de mieux visualiser leur application concrète. C'est également à ce moment qu'on se pose des questions, qu'on tombe sur des problèmes qui ne se poseraient pas forcément en restant dans un tutoriel purement démonstratif des fonctions C++ et QML (des choses du genre « fonctionCpp() » ou encore « foo() » parlent en général très peu).

Au long de cet article, on va décrire le cheminement vers un petit jeu de cartes, le solitaire. Les concepts resteront assez généraux afin de produire rapidement un autre type de réussite et puis pourquoi pas un jeu type « Dame de Pique ». Partant d'une conception orientée objet, on introduira le QML petit à petit.

1.2. Prérequis

Une installation de Qt 4.7 (ou supérieure) fonctionnelle, QtCreator (v 2.2 ou plus). Côté QML, pas besoin de connaissances spécifiques pour démarrer, quelques sauts dans la documentation devraient permettre à tout lecteur de suivre aisément le tutoriel. Si le côté Qt et C++ vous intéresse, une connaissance de QObject, des mécanismes de signaux et slots ainsi qu'une compréhension du pattern modèle-vue-délégué de Qt sont un plus. Les concepts de base ne seront pas réexpliqués, mais le lecteur qui souhaite en savoir plus peut s'appesantir sur ces quelques articles : [Lien 28](#), [Lien 29](#).

1.3. Code source

Le code source du jeu de patience se trouve sur Gitorious :

[Lien 30](#). Un tag est créé avec le nom de chapitre correspondant lorsque c'est renseigné dans l'article, il est donc possible de télécharger et compiler le code en suivant les différentes étapes afin tripoter le code en cours de lecture. Pour ceux qui n'ont pas Git, voici une archive zip du code source aux différentes étapes : [Lien 31](#).

2. Description des moyens de communication entre QML et C++

2.1. Ajout de propriétés au contexte QML

Pour commencer, on va construire le tapis de jeu et afficher le nom de la réussite dessus.

Lancer QtCreator et créer un nouveau projet Qt Quick Application. Choisir un nom pour le projet (par exemple, patience), sélectionner une cible desktop correspondant au système d'exploitation. QtCreator va générer une série de fichiers nécessaires pour lancer une application QML en pleine fenêtre, on peut dès à présent lancer l'application et voir le Hello World apparaître. Voir cet article de Dourouc si vous avez des problèmes pour créer le projet : [Lien 32](#)

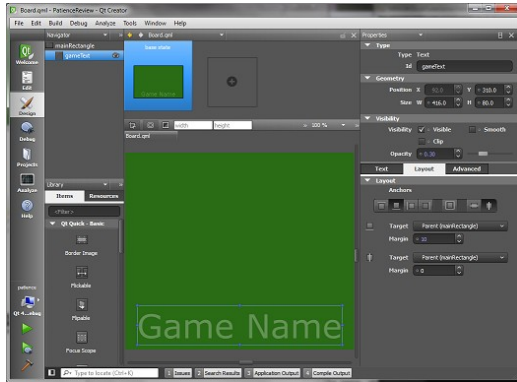
2.1.1. Affichage du tapis de jeu

Tout d'abord on crée un nouveau fichier QML, Board.qml (attention la majuscule est importante !). Un fichier QML commençant par une majuscule met en place un composant réutilisable, que l'on pourra réutiliser à l'envi dans d'autres fichiers QML. Placer ce fichier dans le même répertoire que le fichier main.qml, de manière à être accessible directement. Une fois le fichier ouvert, il est possible de l'éditer en mode texte ou via le QML designer. Les deux sont complètement synchronisés, les modifications effectuées via l'un sont automatiquement répercutées sur l'autre.

Le langage QML est assez intuitif : il s'agit d'un langage déclaratif dans lequel on décrit l'interface, les composants et leur comportement. Il se présente sous la forme d'un arbre d'objets décrits par leurs propriétés.

Ajouter un élément texte, le positionner au centre (via les ancres) et dans le bas du tapis de jeu (board), avec une marge de 50 pixels par exemple. L'élément texte se place sous l'élément board dans l'arbre, il sera donc affiché par-dessus ce dernier. Dans le QML designer il est possible de glisser les éléments d'un niveau à l'autre à l'aide de la souris (l'arbre des objets est affiché en haut à gauche de l'écran). On peut obtenir après quelques manipulations

simples le résultat montré ci-dessous. Les ancrs sont le mécanisme de placement des objets de QML, on décrit simplement comment on positionne un objet par rapport aux autres (par exemple, placer la défaisse à droite de la pile de cartes avec une marge de 50 pixels).



Toutes les fonctions de base sont accessibles dans le designer : on peut choisir la couleur, la police de caractères, l'alignement, etc. C'est assez facile à utiliser et l'avantage est que l'on voit tout de suite l'effet des modifications. Ci-dessous l'équivalent dans l'éditeur texte du code QML. Quelques points qui méritent explication :

- l'attribut id est un attribut réservé, il attribue un nom à notre élément pour y faire référence par la suite ;
- on positionne l'élément texte vis-à-vis du rectangle parent. On décide de le placer en bas, avec une marge de 10 pixels. Il est important de remarquer que le texte est entièrement contraint, positionné via les ancrs. Il est centré horizontalement et attaché par le bas au rectangle. On ne lui fixe pas de taille (via width et height) car elle est déterminée par le texte et la police utilisée ;
- la taille donnée au rectangle vert (600x400) est une taille par défaut. Comme on définit le tapis en tant que nouvel élément réutilisable, il est de bonne pratique d'attribuer une taille par défaut.

```
import QtQuick 1.1

Rectangle {
    id: mainRectangle
    width: 600
    height: 400
    color: "#246612"

    Text {
        id: gameText
        color: "#f3eaea"
        text: "Game Name"
        font.pointSize: 50
        opacity: 0.3
        anchors.bottom: mainRectangle.bottom
        anchors.bottomMargin: 10
        anchors.horizontalCenter:
mainRectangle.horizontalCenter
        font.family: "Verdana"
    }
}
```

Afin d'afficher le nouveau tapis dans l'application, éditer le fichier main.qml et y ajouter l'élément Board. Cela va

simplement le positionner comme seul élément (on va en ajouter d'autres bientôt) de la fenêtre. Démarrer l'application : et voilà, ce n'est pas plus compliqué !

```
import QtQuick 1.1
Board {
}
```

2.1.2. Modification du nom du jeu depuis le C++

Pour modifier le nom de la réussite, on va donner accès à la propriété text de l'élément texte du tapis à l'extérieur du composant. Pour ce faire, on définit un alias dans Board.qml de la manière suivante :

```
Rectangle {
    id: mainRectangle
    property alias gameName: gameText.text
    ...
}
```

On peut maintenant modifier cette propriété depuis main.qml :

```
Board {
    gameName: "nouveau jeu"
}
```

2.1.2.1. Exploration de main.cpp

Avant de modifier le nom depuis le C++, on examine le code généré par QtCreator dans main.cpp afin de mieux cerner le fonctionnement d'une application QML.

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QmlApplicationViewer viewer;
    viewer.setOrientation(QmlApplicationViewer::ScreenOrientationAuto);
    viewer.setMainQmlFile(QLatin1String("qml/patience/main.qml"));
    viewer.showExpanded();
    return app.exec();
}
```

Un objet QmlApplicationViewer est créé, il s'agit de la fenêtre principale du programme, dans laquelle est effectué le rendu QML. Dans Qt4, QmlApplicationViewer est une QDeclarativeView, qui elle-même hérite de QGraphicsView : c'est donc un QWidget. L'orientation du viewer est ensuite spécifiée, ceci n'est utile que pour les applications mobiles qui peuvent être orientées horizontalement ou verticalement. On affiche le widget puis on exécute la boucle d'événements de manière classique. On remarque également la fonction setMainQmlFile qui permet de dire quel fichier QML sert au rendu, on ne doit spécifier que le fichier QML principal, les composants se trouvant dans le même répertoire seront automatiquement trouvés.

2.1.2.2. Contexte

En dessous du capot, le viewer possède un contexte dans lequel va s'exécuter le code QML, on peut le récupérer via viewer.rootContext(); on peut alors y ajouter d'autres variables ou propriétés. Le contexte est le lien entre le C++

et le code QML (voir Communication entre QML et C++/Qt ([Lien 33](#))). Par exemple pour transmettre le nom de la réussite à afficher sur le tapis de jeu, on peut ajouter ceci dans le main.cpp :

```
viewer.rootContext()->
setContextProperty("m_gameName", "Solitaire");
```

Ceci ajoute une propriété `m_gameName` au contexte, dont la valeur est `Solitaire`. Cette propriété est dorénavant accessible depuis le fichier `main.qml` :

```
Board {
    gameId: m_gameName
    ...
}
```

Ainsi on peut injecter des propriétés, des valeurs et même des objets (on le verra plus tard) depuis le C++ vers le QML, le désavantage étant que si la propriété change, il faut refaire un appel à `setContextProperty()`, ce qui n'est pas très pratique.

2.1.3. Un peu de JavaScript dans tout ça

Avec le layout pour positionner le texte, on peut voir que le texte reste centré lorsqu'on redimensionne la fenêtre. Tout se passe très bien, excepté pour le texte, qui reste avec une grosse police quelle que soit la taille de la fenêtre. On peut remédier à cela en introduisant un peu de JavaScript. Dans l'élément texte, on place :

```
font.pixelSize: Math.min(200, parent.width /
text.length * 1.5)
```

Plutôt que d'avoir une taille de pixels fixe pour le texte, on lui donne une taille variable en fonction de divers paramètres que sont la taille de la fenêtre, le nombre de caractères du texte à afficher, etc. Il faut jouer avec la formule pour en voir l'effet, redimensionner la fenêtre et comparer avec une taille fixe.

On peut donc affecter une expression JavaScript à une propriété QML. Cette manière de faire est descriptive et non impérative (au contraire du C++) dans le sens où on décrit comment doit être le texte en fonction d'autres données (largeur de fenêtre ou autre). Ce lien est fort, car si la largeur est modifiée, la taille du texte sera mise à jour automatiquement, c'est un binding.



Le texte se redimensionne automatiquement lorsque la largeur de la fenêtre change.

Les sources de cette partie sont disponibles : [Lien 34](#).

2.2. Ajout d'objets au contexte QML

2.2.1. Modélisation C++ d'une carte

On crée une nouvelle classe `Card`, qui hérite de `QObject`, modèle d'une carte à jouer. Pour le moment le modèle est très simple, on définit les propriétés de la carte via `Q_PROPERTY` :

- suite : cœur, carreau, trèfle ou pique ;
- valeur : de l'as au roi ;
- couleur : rouge ou noir ;
- retourné : si la carte est visible ou non.

```
class Card : public QObject
{
    Q_OBJECT
    Q_ENUMS(Color)
    Q_ENUMS(Suit)
    Q_ENUMS(Value)
    Q_PROPERTY(Color color READ color NOTIFY
colorChanged)
    Q_PROPERTY(Suit suit READ suit NOTIFY
suitChanged)
    Q_PROPERTY(Value value READ value NOTIFY
valueChanged)
    Q_PROPERTY(bool flipped READ flipped() WRITE
setFlipped NOTIFY flippedChanged)
public:
    enum Color { Red, Black };
    enum Suit { Hearts = 0, Diamonds, Clubs,
Spades };
    enum Value { Ace = 1, Two, Three, Four, Five,
Six, Seven, Eight, Nine, Ten, Jack, Queen,
King };
    ...
};
```

Les énumérations sont déclarées avec `Q_ENUMS`, c'est important afin que Qt et QML les reconnaissent comme si elles provenaient de Qt. Les propriétés sont ensuite définies et sont toujours accompagnées d'une clause `NOTIFY` qui renseigne le signal émis lorsque la propriété est modifiée. C'est du ressort du modèle C++ d'émettre ce signal à bon escient. La clause de notification est nécessaire pour maintenir la partie QML et la partie C++ synchronisée, c'est par ce biais que la partie QML saura que le modèle a changé et qu'elle pourra se mettre à jour. Techniquement, ces macros assurent l'indexation dans le système de métaobjets, pour plus de précisions techniques à ce sujet, je conseille la lecture de « De QObject aux métaobjets, une plongée au cœur des fondations de Qt » ([Lien 35](#)).

Les propriétés `couleur`, `suite` et `valeur` sont en lecture seule (pas de méthode `WRITE`), ce qui signifie qu'elles ne pourront pas changer au cours du temps. C'est logique car une carte ne se transforme pas en une autre. Par contre la propriété `flipped` peut être modifiée, lorsqu'on veut retourner la carte par exemple.

Le constructeur initialise la valeur et la suite de la carte :

```
Card::Card(Suit s, Value v, bool flipped, QObject
*parent) :
    m_suit(s),
    m_value(v),
```



```

QObject(parent),
m_flipped(flipped)
{
    m_color = (s == Hearts || s == Diamonds) ?
Red : Black;
}

```

La carte est invisible par défaut et la couleur est déduite de la suite dans le constructeur.

Les variables privées, accesseurs aux propriétés, etc. ne sont pas montrés ici, le code est simple et peut être examiné en détail dans le code source. Deux slots sont créés pour retourner la carte, on prend soin de n'émettre le signal de notification que lorsque l'état de la carte est modifié.

```

void Card::flip() {
    m_flipped = !m_flipped;
    emit flippedChanged(m_flipped);
}

void Card::setFlipped(bool flip) {
    if (m_flipped != flip) {
        m_flipped = flip;
        emit flippedChanged(flip);
    }
}

```

2.2.2. Design QML d'une carte

On ajoute Card.qml au projet, de nouveau avec une majuscule car c'est un composant. Ce composant va dessiner une carte à l'écran. On commence par la définition de quelques propriétés : flipped pour savoir si la carte est retournée ou non, ainsi que la suite et la valeur de la carte. On utilise Item comme élément racine car la carte va être composée de plusieurs sous-éléments. Un item est l'élément de base dont héritent tous les autres items graphiques.

```

Item {
    id: card
    width: 100
    height: 145

    property bool flipped: true
    property int suit: 0
    property int value: 1
}

```

On ajoute ensuite l'élément flipable comme premier composant de la carte, il s'agit d'un composant qui possède un avant et un arrière et qui permet de retourner l'objet, cela convient donc très bien pour une carte à jouer. On utilise anchors.fill : parent afin de spécifier que le flipable prend toute la place disponible dans le parent, c'est-à-dire dans la carte. Sur la face avant, on dessine un rectangle et dans un premier temps, on affiche en texte la valeur et la suite de la carte. La face arrière affiche une image de dos de carte, il suffit de placer le fichier image dans le dossier des fichiers QML.

Un élément QML a une notion de machine d'états : par défaut, il se trouve dans l'état défini par ses propriétés, dont le nom est la chaîne vide. Ici, on définit un état supplémentaire, que l'on nomme back : dans cet état, il

faut tourner la carte de 180 °. Enfin, on indique que l'on se trouve dans l'état back quand la propriété flipped de la carte est vraie. Remarquez la beauté et la concision de l'écriture déclarative.

Chaque item peut également se voir attribuer des transformations géométriques (voir la documentation pour plus de détails sur les possibilités). Ici on définit la rotation de l'élément autour de l'axe y, rotation qui est utilisée pour tourner la carte lorsque l'on passe dans l'état retourné.

Enfin, chaque item peut définir des transitions, il s'agit ici d'animer le retournement afin que celui-ci s'étale sur 500 ms et soit donc visible par l'utilisateur.

```

Flipable {
    id: flipable
    anchors.fill: parent

    front: Rectangle {
        ...
    }

    back: Image {
        anchors.fill: parent
        source: "back.svg"
    }

    states: State {
        name: "back"
        PropertyChanges { target: rotation;
angle: 180 }
        when: card.flipped
    }

    transform: Rotation {
        id: rotation
        origin.x: flipable.width/2
        origin.y: flipable.height/2
        axis.x: 0; axis.y: 1; axis.z: 0 //
set axis.y to 1 to rotate around y-axis
        angle: 0 //
the default angle
    }

    transitions: Transition {
        NumberAnimation { target: rotation;
property: "angle"; duration: 500 }
    }
}

```

Pour tester, il reste une chose à ajouter : une MouseArea pour retourner la carte lorsqu'on clique dessus. On définit cette zone de clic sur toute la carte, et on change simplement la propriété flipped de la carte lorsqu'on clique dessus !

```

Item {
    id: card

    ...

    Flipable {
        ...
    }

    MouseArea {
        anchors.fill: parent
    }
}

```

```

    onClicked: card.flipped = !card.flipped
  }
}

```



Capture d'écran de la carte pendant qu'elle se retourne.

2.2.3. Fournir la carte au QML depuis le C++

setContextProperty peut aussi être utilisée pour ajouter un QObject au contexte : dans ce cas, toutes les propriétés, tous les signaux et tous les slots (plus les méthodes marquées Q_INVOKABLE) sont accessibles depuis le contexte QML.

Il est maintenant assez simple de fournir les cartes depuis le C++ afin que le QML puisse les afficher. On ajoute tout simplement dans le main.cpp les cartes au contexte QML.

```

Q_DECL_EXPORT int main(int argc, char *argv[])
{
    QScopedPointer<QApplication>
    app(createApplication(argc, argv));

    Card* card1 = new Card(Card::Hearts,
Card::Six, true);
    Card* card2 = new Card(Card::Spades,
Card::King, false);

    ...

    viewer.rootContext()-
>setContextProperty("m_card1", card1);
    viewer.rootContext()-
>setContextProperty("m_card2", card2);
    viewer.showExpanded();

    QTimer* timer = new QTimer();
    timer->setInterval(1000);
    QObject::connect(timer, SIGNAL(timeout()),
card1, SLOT(flip()));
    QObject::connect(timer, SIGNAL(timeout()),
card2, SLOT(flip()));
    timer->start();

    return app->exec();
}

```

Pour la démo, on ajoute un petit timer qui va retourner les deux cartes toutes les secondes. Ceci démontre que les propriétés des QObject sont liées au QML, et que lorsqu'elles changent de valeur le système en est averti (grâce au NOTIFY).

Ci-dessous on voit que le code QML de main.qml qui affiche les deux cartes est très simple : on place une carte via Card et on lie les propriétés flipped, suit et value à celles des données de la carte du modèle C++ (m_card).

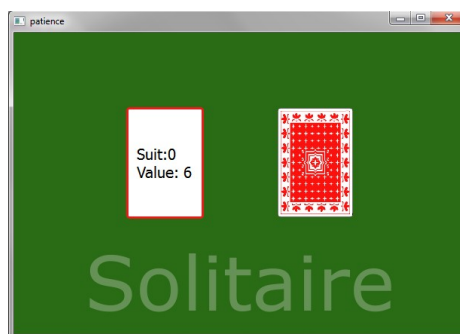
Le désavantage de cette méthode est qu'il faut modifier le code à deux endroits, dans le QML et dans le C++ pour

afficher un nouvel élément. Toute instance d'un objet que l'on veut afficher doit être créée côté C++ au préalable, on verra au point C qu'il existe une autre méthode qui permet au QML d'instancier directement autant d'objets C++ qu'il le désire !

```

Card {
    id: card1
    anchors.top: parent.top
    anchors.topMargin: 100
    anchors.left: parent.left
    anchors.leftMargin: 150
    suit: m_card1.suit
    value: m_card1.value
    flipped: m_card1.flipped
}

```



Les deux cartes se retournent toutes les secondes, reflétant ce qui se passe côté C++.

Vous l'aurez peut-être remarqué, le programme actuel comporte un petit bogue : en effet si on clique sur une carte pour la retourner manuellement, elle s'arrête de tourner toutes les secondes. En fait dans le MouseArea, on casse le binding original en effectuant une affectation directe de la propriété flipped :

```

onClicked: card.flipped = !card.flipped

```

Dans l'application finale, les clics de souris n'agiront plus directement sur l'état des cartes, mais enverront les signaux au modèle C++, qui décidera ce qu'il est bon de faire : de cette manière, on ne casse plus les binding des propriétés. Lien vers les sources du chapitre : [Lien 36](#).

2.2.4. Amélioration du design de la face avant des cartes

Afin de rendre le jeu moins austère, je propose un dessin à la main des cartes. N'étant pas un grand designer ce sera simple. On dispose de quatre images PNG des quatre couleurs de cartes : cœur, carreau, trèfle et pique. Les images sont placées dans le même répertoire que les fichiers QML et devront être distribuées avec les fichiers QML dans l'application finale. On crée un fichier QML supplémentaire, CardFace, qui représente la face d'une carte. Il s'architecture comme suit.

```

Rectangle {
    id: faceRoot
    property int suit: 0
    property int value: 10
    width: 100
    height: 145
}

```

```

color: "#ffffff"
radius: 6
border.width: 1
smooth: true
border.color: "#000000"

function imageSuitPath(suit) {
    if (suit === 0) return "Hearts.png"
    if (suit === 1) return "Diamonds.png"
    if (suit === 2) return "Clubs.png"
    if (suit === 3) return "Spades.png"
}

function valueToText(value) {
    if (value === 1) return "As"
    if (value < 11) return value + '';
    if (value === 11) return 'V';
    if (value === 12) return 'D';
    if (value === 13) return 'R';
}

Component {
    id: cardFaceCorner
    ...
}

```

On commence par un rectangle, auquel on donne les propriétés suite et valeur (qui seront accessibles depuis l'extérieur) ainsi qu'une largeur et une hauteur par défaut et quelques caractéristiques de dessin : arrondis, etc. Ensuite on définit deux fonctions JavaScript utilisées plusieurs fois par la suite, la première donne le nom du fichier image en fonction de la suite de la carte, la seconde le texte à afficher en fonction de la valeur de la carte. On déclare un Component, il s'agit d'un composant QML, comme si on l'avait défini dans un fichier externe, qu'on va pouvoir répéter plusieurs fois. Le but ici est de dessiner le coin supérieur gauche, et de le répéter pour le coin inférieur droit.

```

Component {
    id: cardFaceCorner

    Item {
        Text {
            id: cornerText
            text: valueToText(value)
            anchors.left: parent.left
            anchors.top: parent.top
            font.pixelSize: faceRoot.width / 7
            ...
        }

        Image {
            id: cornerRightImage
            width: Math.min(faceRoot.width,
            faceRoot.height) / 7
            height: Math.min(faceRoot.width,
            faceRoot.height) / 7
            anchors.left: cornerText.right
            anchors.verticalCenter:
            cornerText.verticalCenter
            source: imageSuitPath(suit)
            ...
        }

        Image {

```

```

        id: cornerLowImage
        ...
    }
}

```

Dans les grandes lignes on a : un texte et deux images, le dimensionnement des éléments se fait en fonction des dimensions de la carte (auxquelles on a accès grâce à la définition en ligne du composant). Il est intéressant de remarquer que le composant ne peut avoir qu'un seul item de premier niveau, tout comme dans un fichier QML. C'est pour cela que les trois éléments graphiques sont englobés dans un élément Item.

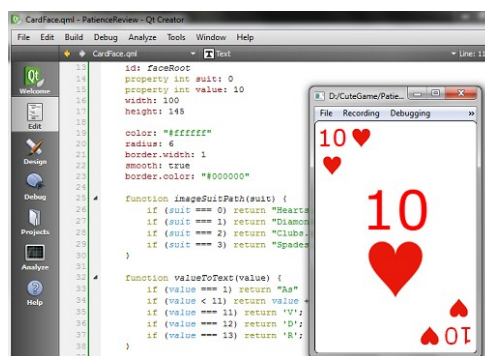
```

Loader {
    sourceComponent: cardFaceCorner
    anchors.top: faceRoot.top
    anchors.left: faceRoot.left
}

Loader {
    sourceComponent: cardFaceCorner
    rotation: 180
    anchors.bottom: faceRoot.bottom
    anchors.right: faceRoot.right
}

```

La déclaration du composant n'affiche rien, il faut instancier les éléments, une fois pour le coin supérieur gauche et une fois pour le coin inférieur droit. Afin de voir l'état d'avancement et surtout de peaufiner les paramètres de rendu (une marge de quatre pixels par-ci, un facteur d'échelle par-là), on peut lancer un aperçu du fichier CardFace.qml avec le QML Viewer. Dans Qt Creator, il suffit de presser F5 lorsqu'on édite le fichier QML, sinon menu Outils - Externe - QtQuick - Aperçu. C'est très pratique, on peut redimensionner la fenêtre, éditer le code (penser à sauvegarder) et faire F5 dans le QML Viewer et le fichier QML se recharge automatiquement. Lien vers les sources de ce Chapitre : [Lien 37](#).



Édition du code QML et visualisation directe du résultat dans le QML Viewer.

2.3. Utiliser les objets directement depuis le QML

Il existe encore une possibilité de lier les objets C++ au code QML de manière très élégante grâce à QObject. On peut instancier toute sous-classe de QObject directement depuis le QML et accéder à toutes les propriétés, tous les signaux et les slots de cet objet. Cette magie est très simple à mettre en place, même pour des classes déjà existantes : dans le main.cpp on ajoute la ligne suivante :

```
qmlRegisterType<Solitaire>("Patience", 1, 0,
"SolitaireModel");
```

Elle enregistre un objet C++ afin qu'il soit reconnu en QML. Patience est le nom de la bibliothèque, suivi des numéros de version et ensuite du nom de composant QML qui correspond au QObject. Le nom ne doit pas forcément être celui de la classe C++, il convient de choisir un nom qui a un sens dans un contexte QML.

Dans le fichier QML, on commence par importer la nouvelle bibliothèque Patience avec le numéro de version renseigné lors de l'appel à qmlRegisterType. Dès cet instant, les nouveaux composants enregistrés dans cette bibliothèque sont disponibles. On peut donc créer un élément SolitaireModel, QtCreator le reconnaît même : ses propriétés et signaux slots sont disponibles en autocomplétion ! Ce qui est élégant, c'est que ce n'est plus le code C++ qui injecte des données dans le code QML comme on l'a vu dans les deux précédentes sections. Ici, le code C++ met des outils à disposition du QML afin d'étendre les capacités. Le code QML a tout le loisir de créer plusieurs solitaires s'il le désire sans que le développeur C++ soit mis à contribution.

```
import QtQuick 1.1
import Patience 1.0

Item {
    ...

    SolitaireModel {
        id: solitaireModel
    }
}
```

Le code du solitaire implémenté ici est une simple démo. On a deux propriétés randomSuit et randomValue, et un slot randomize qui va changer de manière aléatoire la valeur de ces propriétés. Le code QML utilise ces propriétés pour afficher une carte qui, lorsqu'on clique dessus, change aléatoirement. Ci-dessous le code qui implémente cette petite démo.

```
class Solitaire : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name READ name NOTIFY
nameChanged)
    Q_PROPERTY(int randomSuit READ randomSuit
NOTIFY randomSuitChanged)
    Q_PROPERTY(int randomValue READ randomValue
NOTIFY randomValueChanged)

public:
    explicit Solitaire(QObject *parent = 0);

    QString name() const { return "Solitaire"; }
    int randomSuit() const { return m_randomSuit; }
}

int randomValue() const { return
m_randomValue; }

signals:
    void nameChanged(QString name);
    void randomSuitChanged(int arg);
```

```
void randomValueChanged(int arg);
public slots:
    void randomize();
private:
    int m_randomSuit;
    int m_randomValue;
};
Solitaire::Solitaire(QObject *parent) :
    QObject(parent),
    m_randomSuit(0),
    m_randomValue(1)
{
    // Initialize seed for random numbers
    qsrand(QDateTime::currentDateTime().toTime_t(
));
    randomize();
}
void Solitaire::randomize()
{
    m_randomSuit = qrand() % 4;
    m_randomValue = qrand() % 13 + 1;
    emit randomSuitChanged(m_randomSuit);
    emit randomValueChanged(m_randomValue);
}
```

```
Item {
    width: 600
    height: 400
    SolitaireModel {
        id: solitaireModel
    }
    Board {
        anchors.fill: parent
        gameName: solitaireModel.name
        Card {
            id: card1
            anchors.top: parent.top
            anchors.topMargin: 100
            anchors.left: parent.left
            anchors.leftMargin: 150
            suit: solitaireModel.randomSuit
            value: solitaireModel.randomValue
            flipped: false
            onClicked: solitaireModel.randomize()
        }
    }
}
```

Dans le code QML ci-dessus, on utilise le signal onClicked de l'élément Card. Ce signal n'existe pas nativement, il est déclaré dans Card.qml et émis lors du clic sur la carte (via MouseArea).

```
Item {
    id: card
    signal clicked()
    ...
    MouseArea {
        anchors.fill: parent
        onClicked: card.clicked()
    }
}
```

3. Modélisation statique

À la fin de ce chapitre, on aura une application qui affiche la zone de jeu et distribue les cartes. Aucune interaction (déplacement de cartes) ne sera encore possible. Tout est modélisé en C++ mais l'affichage, la partie GUI, sera bien

séparé et effectué en QML. Le QML se limitera à de l'affichage, des animations et des interactions avec le modèle C++ dans lequel se trouve toute la partie logique.

Remarque : cette manière de faire va compliquer la solution pour un simple solitaire, le but de l'article étant de montrer les possibilités du QML, de voir si cette technique serait applicable dans de plus gros projets où l'on ne pourrait se passer du C++ (intelligence artificielle codée en C++, interface d'un logiciel déjà existant...).

3.1. Piles ou listes de cartes

La modélisation d'une pile de cartes, d'un tas, pioche ou défausse nécessite de modéliser une liste de cartes. De manière plus générale, on va décrire les possibilités de QML pour gérer des modèles de données (listes, arbres, etc.).

3.1.1. Première possibilité : un modèle QML

Il est possible de créer un modèle très simplement en QML de la manière suivante :

```
ListModel {
    id: deckModel
    ListElement {
        suit: 1
        value: 5
    }
    ListElement {
        suit: 2
        value: 10
    }
    ListElement {
        suit: 3
        value: 12
    }
}
```

Ceci crée un modèle liste de trois éléments, chacun disposant d'une propriété `suit` et `value`, ce qui permet de caractériser la carte à afficher. Chaque élément doit avoir les mêmes propriétés. Ce modèle n'est cependant pas un modèle C++, on ne s'y attardera donc pas dans cet article. Néanmoins, il peut s'avérer utile pour générer de petits modèles de test lors du prototypage de l'interface QML.

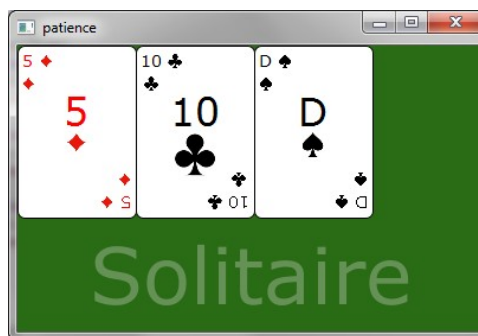
3.1.2. Affichage d'une liste de cartes en QML

Le modèle exposé au paragraphe précédent peut être affiché par le code QML suivant :

```
Row {
    id: row1
    anchors.left: parent.left
    anchors.leftMargin: 0
    anchors.top: parent.top
    anchors.topMargin: 0
    Repeater {
        model: deckModel
        delegate: Card { suit: model.suit; value:
model.value; flipped: false }
    }
}
```

Row est un élément QML qui n'affiche rien mais qui dispose simplement ses enfants en ligne, l'un à côté de

l'autre (il existe l'équivalent `Column`). Chacun des éléments enfants de la ligne est créé par l'élément `Repeater`. Ce dernier prend en paramètres un modèle et un délégué pour l'affichage. Le modèle dans ce cas, est simplement l'identifiant du modèle de la section précédente. Le délégué est l'élément QML qui va être répété par le `Repeater` autant de fois que nécessaire (autant de fois que d'éléments dans le modèle) ; dans ce cas, il s'agit de l'élément `Card` auquel on passe les paramètres `suit` et `value` du modèle. Ci-dessous le résultat de ce code QML (à placer dans le `main.qml` par exemple).



Affichage d'une liste de trois cartes.

3.1.3. Transmission d'une liste d'objets C++

La deuxième possibilité pour afficher une liste de cartes en QML est de transmettre une liste de `QObject*` de la manière suivante (`main.cpp`) :

```
QList<QObject*> cardList;
for (int i = 1; i < 14; i++) cardList.append(new
Card(Card::Clubs, Card::Value(i), true));
viewer.rootContext()->
setContextProperty("m_cardList",
QVariant::fromValue(cardList));
```

Cela transmet la liste d'objets comme valeur de la propriété `m_cardList`, qui peut ensuite être utilisée comme identifiant du modèle QML (dans le `Repeater` de la section précédente). La limitation de cette méthode réside dans le fait que, comme le suggère l'écriture `QVariant::fromValue(cardList)`, une copie de la liste est passée au QML. Cette liste est donc statique, si plus tard on ajoute ou retire d'autres cartes de la liste, le système n'en sera pas averti, à moins d'appeler à nouveau la fonction `setContextProperty`.

3.1.4. Utilisation de `QAbstractListModel`

La troisième solution, la plus puissante, requiert d'implémenter un `QAbstractListModel` du framework modèle/vue de Qt. Cette approche est puissante car elle utilise le modèle Qt déjà connu, cela permet d'utiliser une vue QML ou une vue classique très facilement. Cela requiert néanmoins un peu plus d'effort d'implémentation pour générer le modèle.

Voici le `CardListModel` très basique utilisé, il hérite de `QAbstractListModel` pour plus de facilité :

```
class CardListModel : public QAbstractListModel
{
    Q_OBJECT
public:
    enum CardRoles {
```

```

        SuitRole = Qt::UserRole + 1,
        ValueRole,
        FlippedRole,
        ColorRole
    };
public:
    explicit CardListModel(QObject *parent = 0);
    void pushCard(Card* card);
    void pushCards(QList<Card*> cards);
    Card* popCard();
    int rowCount(const QModelIndex &parent =
    QModelIndex()) const;
    QVariant data(const QModelIndex &index,
    int role) const;
public slots:
    void flipAll();
private:
    QList<Card*> m_cards;
};

```

Le stockage des cartes se fait dans une simple liste de Card*, les fonctions push et pop permettent d'ajouter et retirer des cartes du modèle. Attention de bien suivre les appels aux fonctions du modèle pour tout changement dans celui-ci (comme beginInsertRows) car sans cela la vue QML (ou n'importe quelle autre vue) ne sera pas synchronisée. Consultez la documentation (QAbstractItemModel) pour plus de détails à ce sujet si vous n'êtes pas familier du framework modèle vue de Qt.

On définit plusieurs rôles aux données du modèle, chacun étant une propriété de carte qui sera visible depuis le code QML. La fonction data est surchargée de manière à répondre aux demandes de données concernant ces nouveaux rôles :

```

QVariant CardListModel::data(const QModelIndex
&index, int role) const {
    if (index.row() < 0 || index.row() >
    m_cards.count())
        return QVariant();
    const Card *card = m_cards[index.row()];
    if (role == SuitRole) return card->suit();
    if (role == ValueRole) return card->value();
    if (role == ColorRole) return card->color();
    if (role == FlippedRole) return card-
    >flipped();
    return QVariant();
}

```

Afin que ces rôles nouvellement définis soient visibles depuis QML, il reste à les enregistrer, ce qui est fait dans le constructeur du modèle :

```

QHash<int, QByteArray> roles;
roles[SuitRole] = "suit";
roles[ValueRole] = "value";
roles[ColorRole] = "color";
roles[FlippedRole] = "flipped";
setRoleNames(roles);

```

Enfin le slot flipAll est là pour le show, il retourne chacune des cartes de la liste, et grâce au signal dataChanged, la vue QML sera toujours synchronisée.

```

void CardListModel::flipAll() {
    foreach(Card* card, m_cards) card->flip();
}

```

```

    emit dataChanged(index(0), index(rowCount() -
    1));
}

```

3.2. Jeu du solitaire en C++

Dans cette section, on décrit la modélisation C++ du jeu de solitaire, et on donne les déclarations des objets qui seront utilisés dans le QML par la suite. L'implémentation n'est pas expliquée mais est commentée dans le code. Le jeu de solitaire comprend les éléments suivants :

- la donne : le reste des cartes après distribution ;
- la défausse : pile de cartes retournées depuis la donne ;
- le jeu : les sept piles de cartes du solitaire ;
- les as : les quatre piles de cartes sur lesquelles on monte les as et les autres cartes.

Tout d'abord, retour sur la classe Card qui modélise une carte à jouer (dont l'interface est rappelée ci-dessous). On y trouve les propriétés color, suit et value qui identifient la carte, la propriété flipped qui indique la face à afficher et la propriété selectable qui a été rajoutée. Cette dernière indique si le joueur peut interagir avec la carte (par exemple, dans le contexte du solitaire si la carte est déplaçable). Les propriétés sont en lecture seule, la partie QML ne pouvant pas choisir de retourner une carte ou de la rendre activable, c'est le modèle C++ qui va s'en charger. On définit donc ce qui est visible par QML : les propriétés, les signaux et les slots ainsi que les fonctions Q_INVOKABLE ; de même, ce qui est visible par le C++ : tous les autres membres publics.

```

class Card : public QObject
{
    Q_OBJECT
    Q_ENUMS(Color)
    Q_ENUMS(Suit)
    Q_ENUMS(Value)
    Q_PROPERTY(Color color READ getColor NOTIFY
    colorChanged)
    Q_PROPERTY(Suit suit READ getSuit NOTIFY
    suitChanged)
    Q_PROPERTY(Value value READ getValue NOTIFY
    valueChanged)
    Q_PROPERTY(bool flipped READ isFlipped NOTIFY
    flippedChanged)
    Q_PROPERTY(bool selectable READ isSelectable
    NOTIFY selectableChanged)

public:
    enum Color { Red, Black };
    enum Suit { Hearts = 0, Diamonds, Clubs,
    Spades };
    enum Value { Ace = 1, Two, Three, Four, Five,
    Six, Seven, Eight, Nine, Ten, Jack, Queen,
    King };

public:
    explicit Card(QObject *parent = 0);
    explicit Card(Suit s, Value v, bool isFlipped
    = false, QObject* parent = 0);

    Color getColor() const { return color; }
    Suit getSuit() const { return suit; }
    Value getValue() const { return value; }
    bool isFlipped() const { return flipped; }
    bool isSelectable() const { return

```

```

selectable; }
    bool isValid() const { return value != 0; }

    void flip();
    void setFlipped(bool flip);
    void setSelectable(bool select);

signals:
    void colorChanged(Color);
    void suitChanged(Suit);
    void valueChanged(Value);
    void flippedChanged(bool);
    void selectableChanged(bool);

private:
    Color color;
    Suit suit;
    Value value;
    bool flipped;
    bool selectable;
};

```

La donne et la défausse sont deux listes, deux tas de cartes. Ils sont chacun modélisés par un CardListModel qui hérite de QAbstractItemModel. Ce modèle stocke une liste de Card* et fournit plusieurs fonctions pour la manipulation : construction du paquet de 32 ou 52 cartes, mélange du tas, pioche d'une carte, etc. Ci-dessous, les grandes lignes du code.

```

class CardListModel : public CardModel
{
    Q_OBJECT

public:
    explicit CardListModel(QObject *parent = 0);
    virtual ~CardListModel();

    void pushCard(Card* card);
    void pushCards(QList<Card*> cards);
    Card* popCard();
    QList<Card*> popLasts(int count);
    QList<Card*> takeAll();
    void clear();

    void buildDeck32();
    void buildDeck52();
    void shuffle();

    ...

private:
    QList<Card*> cards;
};

```

Pour la modélisation des sept piles de jeu et des quatre piles d'as, plutôt que d'employer une série de modèles de liste - ce qui est tout à fait envisageable -, on utilise un nouveau modèle avec une structure en arbre à deux niveaux de profondeur. Le premier contient un nœud par pile (soit sept nœuds), le second niveau contient les cartes. Ainsi, on peut mettre en évidence un problème côté QML. En effet, le QML est prévu pour afficher uniquement des modèles de listes à une seule colonne. Il n'est pas prévu de gérer les modèles en arbre directement dans les vues QML (ListView, Column, etc.). Dans le cas du CardListModel, les informations concernant les cartes sont passées via les

rôles, enregistrés par la fonction setRoleNames. Dans le cas du modèle en arbre, on va devoir utiliser un élément QML en particulier afin d'extraire les sous-listes de cartes du modèle et de les afficher dans une ListView, par exemple. Cela est expliqué en détail dans la section Vue en arbre en QML. Le code de CardStackModel, la classe qui modélise les as et le jeu, n'est pas montré ici, son interface est similaire à CardListModel et elle implémente QAbstractItemModel.

Voyons maintenant comme tout ça est mis en musique. La classe Solitaire modélise le jeu en lui-même, initialise le paquet, permet de lancer une nouvelle partie, mélange et distribue, puis autorise les mouvements de carte nécessaires. Toutes ces méthodes nécessaires au QML sont soit des slots, soit notées Q_INVOKABLE.

```

class Solitaire : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QString name READ name NOTIFY
nameChanged)

public:
    explicit Solitaire(QObject *parent = 0);
    QString name() const { return m_gameName; }

    Q_INVOKABLE CardStacksModel* solitaireModel()
const { return m_solitaire; }
    Q_INVOKABLE CardStacksModel* acesModel()
const { return m_aces; }
    Q_INVOKABLE CardListModel* deckModel() const
{ return m_deck; }
    Q_INVOKABLE QAbstractItemModel*
discardModel() const;

signals:
    void nameChanged(const QString&);

public slots:
    void newGame();
    void drawCards();
    void solitaireMove(int startColumn, int
endColumn);
    void discardToSolitaire(int column);
    void discardToAces(int column = -1);
    void solitaireToAces(int solitaireColumn, int
acesColumn = -1);

    ...

private:
    QString m_gameName;
    CardStacksModel* m_solitaire;
    CardStacksModel* m_aces;
    CardListModel* m_deck;
    CardListModel* m_discard;
    NLastProxyModel* m_discardFiltered;
};

```

Le solitaire est composé de deux listes, la donne et la défausse, et de deux arbres, le jeu de solitaire et les as. Ces modèles sont exposés via des méthodes invocables pour l'affichage via QML. Les slots permettent, quant à eux, la manipulation des cartes depuis le QML. À titre d'illustration, voici ci-dessous l'implémentation d'une nouvelle partie : on efface tout, on crée un jeu de 52 cartes, on le mélange et puis on distribue les cartes du solitaire.

```

void Solitaire::newGame() {
    m_solitaire->clearAllStacks();
    m_aces->clearAllStacks();
    m_deck->clear();
    m_discard->clear();

    m_deck->buildDeck52();
    m_deck->shuffle();

    for (int i = 0; i < m_solitaire-
>stacksCount(); ++i) {
        for (int j = i; j < m_solitaire-
>stacksCount(); j++) {
            Card* card = m_deck->popCard();
            if (card == 00) {
                qWarning() << "Not enough cards
in the deck to complete distribution";
                return;
            }
            if (i == j) {
                card->flip();
                card->setSelectable(true);
            }
            m_solitaire->pushCard(j, card);
        }
    }
}

```

Le code complet se trouve dans les sources, on y trouve également le code de la réussite Freecell, qui s'implémente dans la classe Freecell (elle gère la logique de la partie de Freecell). Elle utilise également les modèles décrits précédemment de liste et d'arbre, ce qui permet d'écrire cette classe très rapidement.

Pour clore cette partie, un petit mot pour signaler l'existence de tests unitaires. Ces tests valident le modèle C++ qui est proposé. Ils sont totalement indépendants de l'interface graphique, que celle-ci soit écrite à base de QWidgets ou de QML. Ces tests font partie d'un projet séparé (un autre .pro car il y a un autre exécutable) qui se trouve également sur le dépôt : [Lien 38](#).

3.3. Affichage du Solitaire en QML

3.3.1. Mockup - disposition des éléments

Pour commencer, on réfléchit à la disposition des différents éléments sur l'écran les uns par rapport aux autres. On veut dessiner la donne et la défausse en haut à gauche, les as en haut à droite et le solitaire en bas de tout ça. Schématiquement, cela donne un dessin relativement flashy :



Disposition schématique.

Grâce aux ancres QML, il est très facile de produire ce positionnement : il suffit d'indiquer à chaque rectangle où il se trouve par rapport aux autres éléments. On peut en voir l'effet très rapidement dans QtCreator via QML Viewer ou le designer.

```

Rectangle {
    property int globalMargin: 30

    height: 600
    width: 800

    Rectangle {
        id: deck
        anchors.left: parent.left
        anchors.top: parent.top
        anchors.leftMargin: globalMargin
        anchors.topMargin: globalMargin

        ...
    }

    Rectangle {
        id: discard
        anchors.left: deck.right
        anchors.top: parent.top
        anchors.topMargin: globalMargin
        anchors.leftMargin: globalMargin

        ...
    }

    Rectangle {
        id: aces
        anchors.right: parent.right
        anchors.top: parent.top
        anchors.topMargin: globalMargin
        anchors.rightMargin: globalMargin

        ...
    }

    Rectangle {
        id: solitaire
        anchors.top: aces.bottom
        anchors.left: parent.left
        anchors.right: parent.right
        anchors.bottom: parent.bottom
        anchors.margins: globalMargin

        ...
    }
}

```

3.3.2. Réalisation

On applique maintenant les techniques décrites précédemment pour construire, bloc par bloc, le solitaire. Pour commencer, on met l'élément Board en fond et on crée l'élément SolitaireModel, ce qui va instancier la classe Solitaire côté C++. On récupère alors la propriété name pour la fournir à l'élément Board. Voici à quoi ressemblent Solitaire.qml et Main.qml.

```

main.qml
Item {
    width: 800

```



```

height: 600

Solitaire {
    anchors.fill: parent
}

```

Solitaire.qml

```

Item {
    property int globalMargin: 30

    SolitaireModel {
        id: solitaireModel
    }

    Board {
        anchors.fill: parent
        gameName: solitaireModel.name

        Rectangle {
            id: deck
            ...
        }

        Rectangle {
            id: discard
            ...
        }

        Rectangle {
            id: aces
            ...
        }

        Rectangle {
            id: solitaire
            ...
        }
    }
}

```

Petite remarque : main est le fichier QML principal, celui qui est exécuté par l'application. Dans ce cas, width et height donnent les dimensions par défaut de la fenêtre qui sera créée. Par contre pour le composant Solitaire, on ne donne ni dimensions ni ancrage (dans Solitaire.qml) car c'est un composant réutilisable, c'est à celui qui utilise Solitaire de le positionner et de le dimensionner, comme on le fait avec l'élément Rectangle. De fait on positionne et dimensionne le Solitaire depuis main.qml avec la ligne anchors.fill : parent, ce qui a pour effet que le solitaire prend toute la place disponible dans la fenêtre. On verra plus loin pour améliorer le design afin de pouvoir afficher aussi le Freecell.

3.3.3. Vue du deck et de la défausse

Le deck et la défausse sont basés sur le modèle de liste, on va donc les afficher avec un nouveau composant QML : CardListView.qml. Il affiche les cartes côte à côte, et dispose des propriétés horizontalSpacing et verticalSpacing pour choisir d'afficher en ligne ou en colonne (ou en diagonale). Lorsqu'il n'y a pas de carte à afficher (modèle vide) on affiche un rectangle afin de représenter visuellement l'emplacement vide de la pile. Ce rectangle est totalement recouvert par la première carte de la liste. Ci-dessous le code simplifié de la vue QML. On

utilise un simple Repeater pour gérer la création et la destruction des délégués (Card.qml). On stipule les propriétés x et y pour en contrôler la disposition. Note : le property alias donne simplement accès au modèle du répéteur depuis l'extérieur du composant.

```

Item {
    id: root
    property alias cardListModel :
cardListView.model
    property int horizontalSpacing: 25
    property int verticalSpacing: 25

    signal cardClicked(int index, int suit, int
value)

    width: cardsWidth + horizontalSpacing *
Math.max(0, (cardListView.count - 1))
    height: cardsHeight + verticalSpacing *
Math.max(0, (cardListView.count - 1))

    Rectangle {
        anchors.left: parent.left
        anchors.top: parent.top
        anchors.leftMargin: 2
        anchors.topMargin: 2
        width: cardsWidth - 4
        height: cardsHeight - 4
        radius: 5
        border.width: 3
        color: "transparent"
        border.color: "black"

        MouseArea {
            anchors.fill: parent
            onClicked: root.cardClicked(-1, -1,
-1);
            onDoubleClicked:
root.cardDoubleClicked(-1, -1, -1);
        }
    }

    Repeater {
        id: cardListView

        delegate: Card {
            id: thiscard
            x: horizontalSpacing * index
            y: verticalSpacing * index
            value: model.value;
            suit: model.suit;
            flipped: model.flipped;
            onClicked: root.cardClicked(index,
suit, value)
        }
    }
}

```

À présent, on remplace dans le mockup les rectangles de la pioche et de la défausse par CardListView. Sans changer quoi que ce soit aux ancres, on renomme les Rectangle par des CardListView et on ajoute les propriétés qui vont bien. Afin de démontrer le bon fonctionnement, on remonte le signal clicked sur le deck et on appelle le slot drawCards du solitaire. Le code QML reste élégamment simple. Le code C++ appelé met à jour les modèles sous-jacents et les vues se tiendront automatiquement synchronisées.

```

CardListView {

```

```

id: deck
cardListModel: solitaireModel.deckModel()
horizontalSpacing: 0
verticalSpacing: 0

anchors.left: parent.left
anchors.top: parent.top
anchors.leftMargin: globalMargin
anchors.topMargin: globalMargin

MouseArea {
    anchors.fill: parent
    onClicked: { drawCards(); }
}

CardListView {
    id: discard
    cardListModel: solitaireModel.discardModel()
    horizontalSpacing: 30
    verticalSpacing: 0

    anchors.left: deck.right
    anchors.top: parent.top
    anchors.topMargin: globalMargin
    anchors.leftMargin: globalMargin
}

```

Afin que tout ceci fonctionne, il faut enregistrer les classes modèles dans le main.cpp comme ci-dessous. Sans cela, rien ne s'affiche.

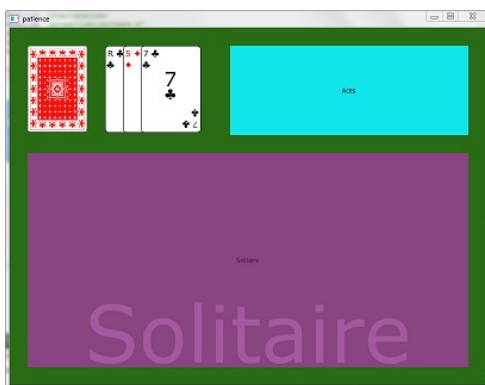
```

qmlRegisterType<Solitaire>("Patience", 1, 0,
"SolitaireModel");

qmlRegisterType<CardListModel>();
qmlRegisterType<QAbstractItemModel>();
qmlRegisterType<CardStacksModel>();

```

Ci-dessous, le résultat, exécuter l'application et cliquer sur la pioche pour voir les cartes défiler dans la défausse. Une fois la pioche vide, c'est la défausse qui est rechargée sur le tas. Remarque : on ne voit toujours que les trois dernières cartes dans la défausse, cela est voulu dans le jeu du solitaire (celui de mon grand-père en tous cas :p). En fait si on examine le code C++, on voit que le modèle de la défausse renvoyé est un proxy sur la défausse totale, proxy qui filtre les cartes pour n'afficher que les trois dernières. Lien vers les sources de la section : [Lien 39](#).



CardListView en action.

3.3.4. Le solitaire et les as : la vue arbre en QML

Les vues QML ne sont prévues que pour des modèles de

liste : dès que le modèle devient plus compliqué, il faut ruser. C'est ici qu'intervient l'élément QML « VisualDataModel », assez compliqué ; on va essayer de le décrire au mieux. Tout d'abord, cet élément, comme son nom semble l'indiquer, est un modèle : il peut être utilisé partout où un modèle est demandé. Sa particularité est d'être visuel : en fait, il encapsule également le délégué (dans la terminologie Qt). C'est donc une espèce de modèle qui fournit des données mais aussi le délégué pour les visualiser.

Dans le cas présent, le VisualDataModel a une propriété très intéressante : on peut lui assigner un nœud du modèle via un QModelIndex, il se comporte alors comme la liste des enfants de ce nœud. De cette manière, il est possible d'itérer récursivement dans la profondeur d'un modèle en arbre de n'importe quel type (QAbstractItemModel). Afin d'afficher les colonnes du jeu, on crée un composant CardStackView, qui affiche les colonnes les unes à côté des autres ; pour afficher les colonnes, on réutilise le CardListView vu au paragraphe précédent.

Pour commencer, on modifie CardListView de manière à utiliser un VisualDataModel : pour la liste, cela ne change rien, car le nœud utilisé par défaut est la racine. Dans le code de CardListView, on donne accès aux propriétés model et rootIndex à l'extérieur du composant. On remplace le modèle qui était attribué directement par un composant VisualDataModel auquel on attribue le model et le rootIndex.

```

Item {
    id: root
    property alias cardListModel:
cardListDataModel.model
    property alias cardListModelIndex:
cardListDataModel.rootIndex

    ...

    Repeater {
        id: cardListView
        model: VisualDataModel {
            id: cardListDataModel

            delegate: Card {
                ...
            }
        }
    }
}

```

Ensuite, du côté de CardStackView, on crée une ligne dans laquelle on répète une CardListView. On utilise aussi ici un VisualDataModel, simplement pour avoir accès à la fonction modelIndex(int) qui renvoie un QModelIndex que l'on attribue au rootIndex de CardListView. Le code est somme toute assez simple, expressif et bien réutilisé.

```

Item {
    id: root
    property QObject cardStackModel

    ...

    Row {

```

```

spacing: horizontalSpacing

Repeater {
    id: stackRow
    model: VisualDataModel {
        id: visualModel
        model: cardStackModel
        delegate: CardListView {
            horizontalSpacing: 0
            verticalSpacing:
root.verticalSpacing

            cardListModelIndex:
visualModel.modelIndex(index)
            cardListModel: cardStackModel
        }
    }
}
}
}
}

```

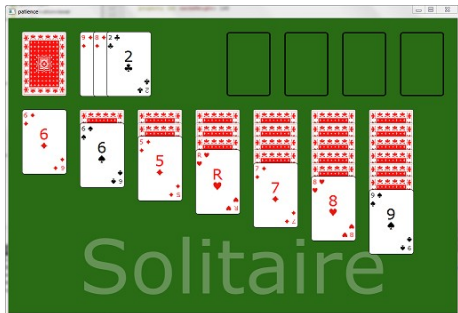
```

Item {
    id: root
    property int globalMargin: Math.max(10, width
/ 50)
    property int horizontalSpacing: width / 50
    property int cardsWidth: (root.width - 2 *
root.globalMargin - 6 * root.horizontalSpacing) /
7
    property int cardsHeight: cardsWidth * 1.45
    ...

    CardListView {
        ...
        cardsWidth: root.cardsWidth
        cardsHeight: root.cardsHeight
        ...
    }
    ...
}

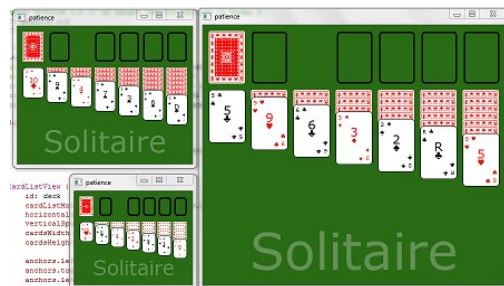
```

Il reste bien sûr à remplacer les rectangles du mockup, par les CardStackView, ce qui se fait aisément comme pour les listes. Voici le résultat actuel, le jeu commence à prendre forme.



Placement des CardStackView pour le jeu de solitaire et pour les piles d'as.

La propriété cardsWidth est calculée sur la base de la largeur de l'élément racine (Solitaire.qml) selon la formule ci-dessus, formule qui découle du comptage de ce qu'on trouve sur une largeur de solitaire. La hauteur de la carte est proportionnelle afin de garder toujours le même facteur de forme. Enfin, on affecte à chaque CardListView et CardStackView les dimensions calculées.



On peut redimensionner la fenêtre, les cartes se redimensionnent.

3.4. Mise à l'échelle

Comme on peut le constater en redimensionnant l'application, tout ne se comporte pas au mieux. On ne voit pas toujours toutes les cartes, les cartes sont soit trop grandes soit trop petites...

Pour que tout se redimensionne au mieux, on va exprimer les contraintes suivantes en QML :

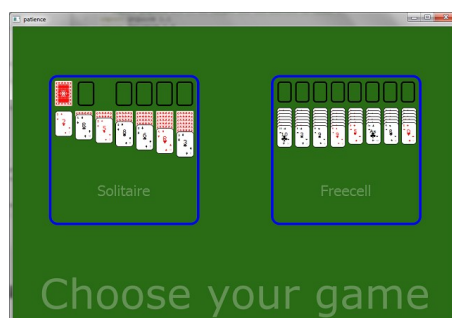
1. Toutes les cartes doivent avoir la même taille sur le plateau de jeu ;
2. La plus grande dimension est celle des sept colonnes de cartes en largeur, elle doit prendre toute la largeur disponible.

Pour le premier point, on a déjà bien préparé le travail : en effet, si on regarde les éléments CardListView, CardStackView et Card, on y a placé les propriétés cardWidth et cardHeight qui permettent de forcer la taille des cartes. Quelle taille leur donner ? C'est ici que le second point intervient : on veut maximiser la largeur des cartes de manière à occuper tout l'espace disponible. Il suffit de compter ce qu'on trouve sur la largeur du solitaire : deux marges, sept largeurs de cartes, six espacements entre cartes. D'où le code suivant dans Solitaire.qml :

3.5. Bonus : le jeu de Freecell et un menu

Pour prouver la bonne conception et la bonne réutilisation des éléments QML, on implémente le design du Freecell. Ce dernier étant similaire au solitaire, on laisse au lecteur intéressé le soin de faire l'exercice.

Chose plus intéressante, réaliser un petit menu, un moyen de choisir au démarrage de l'application quelle réussite on veut jouer. Voici ce que l'on propose de réaliser : le solitaire et le Freecell sont affichés côte à côte ; lorsqu'on clique sur l'un, il passe en plein écran. La touche Escape permet de revenir au menu.



Le menu permet de choisir la patience à jouer.

Pour ce faire, on change quelque peu l'architecture dans le QML. Plutôt que d'avoir un Board (tapis vert) dans le solitaire et un dans le Freecell, on place un seul Board dans le main sur lequel on dispose les deux patiences. Ensuite on va utiliser la notion d'état en QML. On définit trois états : l'état par défaut est le menu en quelque sorte, tel que montré ci-dessus ; un second état est créé lorsque le solitaire est en plein écran ; et un troisième pour le Freecell. Dans le code ci-dessous, SelectionRectangle est un rectangle dessiné par-dessus le jeu afin d'en délimiter l'espace visuellement (cadre rouge ou bleu) dans le menu.

```
Item {
    id: root
    width: 900
    height: 600

    Board {
        id: board
        anchors.fill: parent
        gameId: "Choose your game"

        SelectionRectangle {
            id: solitaireRectangle
            gameId: solitaire.gameName

            ...

            onClicked: root.state = "Solitaire"
        }
        Solitaire {
            id: solitaire
            anchors.fill: parent
        }
    }

    SelectionRectangle {
        id: freecellRectangle
        gameId: freecell.gameName

        ...

        onClicked: root.state = "Freecell"
    }
    Freecell {
        id: freecell
        anchors.fill: parent
    }
}

states: [
    State {
        name: "Solitaire"
        AnchorChanges {
            target: solitaireRectangle
            anchors.horizontalCenter:
undefined
            anchors.top: board.top
            anchors.left: board.left
            anchors.right: board.right
            anchors.bottom: board.bottom
        }
        PropertyChanges {
            target: solitaireRectangle
            anchors.topMargin: 0
            fullscreen: true
        }
    }
]
```

```

    }
    PropertyChanges {
        target: board
        gameId: solitaire.gameName
    }
    PropertyChanges {
        target: freecellRectangle
        opacity: 0
    }
},
State {
    name: "Freecell"
    ...
}

transitions: Transition {
    AnchorAnimation { duration: 300;
easing.type: Easing.InOutQuad }
    NumberAnimation { target:
solitaireRectangle; property: "opacity";
duration: 200; easing.type: Easing.InOutQuad }
    NumberAnimation { target:
freecellRectangle; property: "opacity"; duration:
200; easing.type: Easing.InOutQuad }
}

focus: true
Keys.onPressed: {
    if (event.key == Qt.Key_Escape)
root.state = ""
}
}
```

Qu'y a-t-il d'intéressant dans ce code? Tout d'abord, la déclaration des états Solitaire et Freecell. On y déclare tout ce qui change par rapport à l'état initial : un ancrage plein écran, plus de marges, le nom à afficher sur le plateau et on met l'autre réussite invisible. Le changement d'état se fait d'un clic sur la zone de prévisualisation simplement par cette déclaration `onClicked: root.state = 'Solitaire'`. Le retour au menu se fait via le clavier, on assigne alors à l'état une chaîne de caractères vide, ce qui indique un retour à l'état par défaut.

Enfin, pour animer le tout, on utilise des transitions. Les transitions définissent des animations lors d'un changement d'état. On anime ici le changement d'ancres et le changement d'opacité. Voir la documentation pour connaître tous les paramètres de ces animations (en boucle, en parallèle, différentes courbes, etc.).

Ceci termine le chapitre concernant le dessin du jeu. Seulement, il reste encore à faire vivre ce jeu, à gérer l'interaction avec l'utilisateur. Cela fait l'objet du chapitre 4, qui est donc plus orienté QML. L'interaction avec le C++ se limitant à appeler les slots du modèle de jeu. Lien vers le code final : [Lien 40](#).

4. Interactions utilisateur

Jusqu'ici rien ne bouge, pas moyen de déplacer les cartes et de faire avancer la réussite ! Cela sera abordé dans un autre article prochainement.

Retrouvez l'article de Stéphane Fabry en ligne : [Lien 41](#)

Le guide de développement d'application de bureau avec Qt Quick

Le but de ce guide est de vous familiariser avec Qt Quick et QML en employant les meilleures pratiques de programmation. Notez qu'il est tout de même nécessaire de connaître les bases de QML. À travers ce guide, on verra quelques bonnes techniques à utiliser. On terminera par voir comment déployer son application sur les plateformes de bureau. N'hésitez pas à consulter le code source fourni avec ce guide pour mieux comprendre les exemples et plus généralement la programmation avec Qt Quick !

1. L'article original

Cet article est une adaptation en langue française de Qt Quick Application Developer Guide for Desktop, Release 1.0 ([Lien 42](#)).

2. À propos de ce guide

2.1. Pourquoi lire ce guide ?

Ce guide apporte un aperçu des fonctionnalités de Qt Quick et de QML concernant le développement d'applications de bureau.

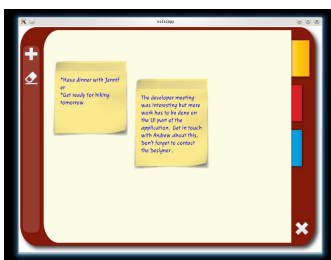
On se concentrera sur Qt Quick et les façons de l'utiliser efficacement pour écrire des applications complètes sans aucune ligne de code en C++. Il vous guidera pas à pas, des réglages de l'environnement de développement à la création de projets, jusqu'à la phase finale : le déploiement. On créera une application de gestion des notes à l'aide de Post-it, « NoteApp ».

Chaque chapitre est divisé en plusieurs étapes. Chacune décrit une fonctionnalité particulière de l'application et le code QML nécessaire. Ce guide couvre plusieurs aspects d'une interface utilisateur moderne tels que les animations, les bases de données et l'utilisation de JavaScript pour la logique de l'application.

Cette application ne ressemblera pas à une application de bureau classique. On ne retrouvera pas les éléments typiques tels que des barres d'outils, des menus, des dialogues, etc. Elle est fortement inspirée des interfaces modernes et fluides que l'on pourrait retrouver sur tablette ou smartphone, alors qu'elle cible plus un environnement de bureau.

Pour faciliter la lecture, chaque chapitre propose une version de NoteApp. Vous devriez vous y référer au cours de la lecture de ce guide. Les sources sont disponibles : [Lien 43](#).

À la fin de ce guide, vous devriez bien comprendre comment fonctionne la création d'applications avec QML et Qt Quick, tout en ayant déjà pris de bonnes habitudes.



Une capture d'écran de NoteApp.

2.2. Et ensuite ?

On commence par créer un prototype de NoteApp. On découvre ainsi comment QML peut aider pendant cette étape.

3. Design de base et prototypes

L'un des principaux avantages de Qt Quick et de QML est qu'ils permettent de rapidement parvenir à un prototype. La phase de prototypage sera donc la première étape, et ce pour deux raisons. Premièrement, les designers peuvent rapidement et facilement parvenir à un premier jet de l'interface comme expliqué plus haut. Deuxièmement, cette phase incite les développeurs à travailler main dans la main avec les designers. Cette relation développeur/graphiste permet de développer rapidement et sûrement.

Plus tard, ce prototype sera utilisé comme base lorsque le développement commencera réellement. Dans ce chapitre, on avancera pas à pas en commençant par voir quelques concepts faits à la main. On discutera des fonctionnalités désirées et de la façon dont doit réagir l'interface aux interactions de l'utilisateur.

On abordera quelques concepts de base de QML tels que la création de composants simples ainsi que la disposition de ces derniers.

Note : vous pouvez retrouver les codes relatifs à ce chapitre dans le dossier compressé disponible dans l'introduction.

Voici un bref résumé de l'objectif de ce chapitre :

- concepts de l'interface et discussion sur les fonctionnalités ;
- création de composants QML avec Qt Creator ;
- utilisation de la propriété *anchor* et des éléments *Repeater* pour disposer correctement les composants QML.

3.1. Aperçu de l'application NoteApp

L'application *NoteApp* utilise un design basé sur l'utilisation de Post-it pour la création de notes et leur enregistrement.

Pour simplifier l'organisation de ses notes, l'utilisateur devra pouvoir les ranger dans des catégories prédéfinies. On imagine donc avoir trois catégories auxquelles elles peuvent appartenir. Chaque catégorie sera représentée par une zone dans laquelle les notes seront placées. On y ajoute donc aussi l'élément *Page*, une *Page* étant une zone dans laquelle les notes seront créées et placées.

L'utilisateur pourra supprimer une note à la fois ou toutes en même temps et les déplacer librement sur la page. Les trois pages correspondront aux trois catégories et seront repérées par des signets. Chaque signet possèdera une couleur particulière.

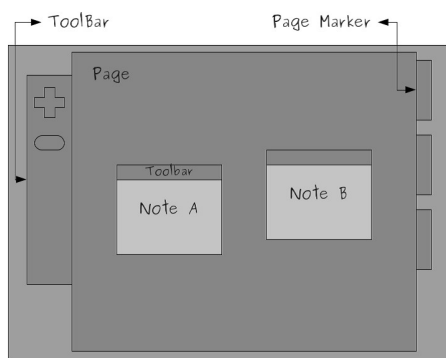
Ce serait aussi une bonne idée de stocker les notes localement et peut-être de les enregistrer automatiquement pour l'utilisateur.

Récapitulatif :

- création/suppression des *Notes* ;
- édition et positionnement libre sur la page ;
- stockage local ;
- séparation en trois pages indiquées par des signets.

3.1.1. Éléments de l'interface

On commence par implémenter le design de l'image du dessous.



Elle donne une bonne idée de ce que pourrait imaginer l'utilisateur concernant une telle interface. Elle permet également d'identifier les différents éléments et interactions à implémenter plus tard.

3.1.2. Comportement de l'interface

Comme dit précédemment, trois pages peuvent contenir des items *Note*. On peut aussi constater que les signets pour changer de page sont placés sur la droite. La barre d'outils de gauche ne contient que deux boutons : un pour l'ajout et l'autre pour la suppression de toutes les notes de la page. Chaque item *Note* possède lui-même une barre d'outils qui lui permet d'être déplacé à l'aide d'un système de glisser-déposer. Sur chaque note, un bouton permet de supprimer la note.

3.1.2.1. Et ensuite ?

On tente d'identifier les composants nécessaires à l'implémentation des fonctionnalités qu'on a choisies et apprendre à les créer.

3.2. Création de composants QML pour l'interface utilisateur

Une fois les fonctionnalités et le design d'interface utilisateur pour l'application *NoteApp* choisis, on peut commencer sans problème à implémenter un premier jet.

Le prototype sera très simpliste et ne contiendra aucune fonctionnalité, mais donnera une idée de la version finale

de l'application.

Dans cette étape, on crée notre première interface avec *Qt Quick* en utilisant *Qt Creator* et on verra comment créer les premiers composants QML.

3.2.1. Création d'un projet UI Qt Quick dans Qt Creator

L'utilisation de *Qt Creator* simplifie vraiment la vie pendant la phase de prototypage. Vous vous en rendrez sûrement compte rapidement. C'est la façon la plus efficace, surtout si vous n'arrêtez pas de changer, tester et bidouiller dans chaque composant QML. N'hésitez pas à regarder les effets d'une modification, même mineure, cela vous permettra d'identifier plus facilement les erreurs.

Pour comprendre comment créer un projet d'interface *Qt Quick*, la documentation propose quelques pistes : [Lien 44](#).

Note : un unique fichier QML « *main.qml* » au début sert à charger et à lancer l'application. Pour *NoteApp*, le fichier *main.qml* a en réalité été généré automatiquement par *Qt Creator*.

3.2.2. Créer des composants QML à partir d'éléments plus simples

Pour rapprocher la programmation orientée objet de QML, on pourrait comparer les composants QML à des classes utilisées pour instancier et déclarer des objets. On pourrait en fait écrire entièrement une application simple dans un gros fichier QML, mais cela augmenterait sûrement la complexité et empêcherait de réutiliser le code.

Un composant QML peut être vu comme un groupe d'éléments QML, bien qu'en général il n'en contienne qu'un seul.

Note : chaque composant QML est contenu dans son propre fichier QML (.qml) éponyme. Par exemple, le composant *Note* sera dans un fichier appelé *Note.qml*.

D'après le concept d'interface, voici une brève liste des composants qui s'imposent d'eux-mêmes. On en ajoutera d'autres plus tard, lorsque l'on reverra l'application.

- *Note* : représente un item note.
- *Page* : contient des notes.
- *Marker* : représente un signet de page, permet à l'utilisateur de changer de page.
- *NoteToolbar* : la barre d'outils présente sur chaque note qui permet les déplacements avec la souris.

Jetez un coup d'œil à la page de la documentation dédiée ([Lien 45](#)) pour comprendre comment utiliser *Qt Creator* pour créer les composants dont on a parlé. On regardera en détail la création de chacun d'entre eux dans les chapitres à venir.

3.2.2.1. Et ensuite ?

On commence à créer le prototype de l'interface de l'utilisateur.

3.3. Manipuler la disposition des items QML et créer ses propres composants

L'élément graphique « Rectangle » est un choix qui paraît évident pour créer les composants de base du prototype. Ils sont très faciles à utiliser et à mettre en place.

Note : c'est toujours une bonne idée d'indiquer les dimensions de base d'un composant tout juste créé. Tester l'application devient alors beaucoup plus facile.

On commence par implémenter le composant *Note*.

3.3.1. Note et NoteToolbar

On a normalement créé les fichiers QML nécessaires avec Qt Creator.

Le code de NoteToolbar devrait ressembler à ceci pour correspondre à nos attentes :

```
// NoteToolbar.qml
import QtQuick 1.1

// Élément Rectangle avec dimensions et couleurs
prédéfinies.
Rectangle {
    id: root
    width: 100
    height: 62
    color: "#9e964a"
}
```

Le composant *Note* possédera un composant *NoteToolbar*. De plus, il disposera d'un élément de saisie pour récupérer le texte entré par l'utilisateur. On utilisera l'élément QML *TextEdit* à cet effet. Pour arranger ces items entre eux, on utilise la propriété *anchor*. Cette propriété provient de l'élément *Item*, dont toutes les classes QML dérivent.

Référez-vous à la documentation pour plus de détails sur l'utilisation de la propriété *anchor* : [Lien 46](#).

```
// Note.qml
import QtQuick 1.1
Rectangle {
    id: root
    width: 200
    height: 200
    color: "#cab1b"

    // Création de l'item NoteToolbar qui
    sera positionné/attaché à son parent.
    NoteToolbar {
        id: toolbar
        // La hauteur doit être spécifiée
        // puisque'il
        // n'y a pas de positionnement
        // relatif sur le bas.
        height: 40

        // " Liaison " au parent grâce
        // aux
        // positionnements relatifs sur
        // le haut,
        // la gauche et la droite.
        anchors {
            top: root.top
```

```
        left: root.left
        right: root.right
    }
}

// Création de la zone de saisie.
TextEdit {
    anchors {
        top: toolbar.bottom
        bottom: root.bottom
        right: root.right
        left: root.left
    }
    wrapMode: TextEdit.WrapAnywhere
}
```

Note : on ne peut pas combiner l'utilisation de la propriété *anchor* avec un positionnement absolu (ce qui semble logique).

Attention, pour des raisons de performances, vous ne devriez pas utiliser la propriété *anchor* avec des items qui n'ont pas de lien direct avec cet item (c'est-à-dire l'item parent et les items « frères », qui ont le même parent).

3.3.2. Page

Une fois le composant *Note* prêt, on essaye d'en mettre quelques-uns dans un composant *Page*.

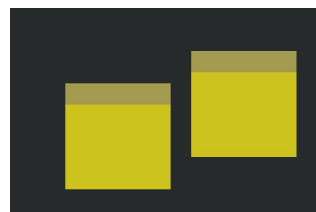
```
// Page.qml
import QtQuick 1.1

Rectangle {
    id: root
    width: 600
    height: 400
    color: "#222525"

    // Création d'une Note.
    Note {
        id: note1
        // Les propriétés x et y servent
        // à définir un positionnement
        // absolu.
        x: 105; y: 144
    }

    Note {
        id: note2
        x: 344
        y: 83
    }
}
```

Avec *Qt Creator*, vous pouvez simplement exécuter ce fichier. Il lancera alors *qmlviewer* qui se chargera d'afficher le fichier *Page.qml*. Le résultat devrait ressembler à ceci :



3.3.3. Signets (Marker)

Comme pour chaque autre composant, *Marker* sera simplement un rectangle avec des dimensions prédéfinies. Son comportement sera défini plus tard.

```
// Marker.qml

import QtQuick 1.1

Rectangle {
    id: root
    width: 50
    height: 90
    color: "#0a7bf8"
}
```

3.3.3.1. Et ensuite ?

Dans la prochaine partie, on verra comment utiliser les éléments *Repeater* et *Column* pour gérer une liste statique de signets.

3.4. Utilisation d'un Repeater et d'un délégué pour créer la liste de signets

Précédemment, on a vu comment créer les composants *Note*, *NoteToolbar*, *Page* et *Marker* et comment les positionner à l'aide de la propriété *anchor*.

Pourtant, si on regarde le concept, on remarque la présence de trois éléments *Marker* (signets) arrangés verticalement. On pourrait de nouveau utiliser la propriété *anchor*, mais cela augmenterait largement la complexité du code. QML dispose de moyens bien plus efficaces et propres tels que les dispositions (*layout*) et autres éléments de positionnement comme *Column*.

Comme on veut trois signets presque identiques dans un unique élément *Column*, on utilise l'élément *Repeater*, qui les dupliquera.

```
Column {
    id: layout
    // La propriété spacing peut être
    utilisée pour ajouter de l'espace entre les
    items.
    spacing: 10

    // Un Repeater pour générer trois items
    Marker.
    Repeater {
        model: 3
        delegate:
        // Utilisation de Marker comme
        délégué.
        Marker { id: marker }
    }
}
```

Dans ce bout de code, l'élément *Repeater* génère trois items (*model: 3*) d'après le délégué *delegate*, l'élément *Marker*.

Vous pouvez trouver d'autres informations sur l'utilisation conjointe des éléments *Column* et *Repeater* : [Lien 47](#).

Vous vous demandez sûrement maintenant où placer le

code du dessus. Eh bien, on doit le mettre dans un autre composant : *MarkerPanel*. En fait, il s'agit simplement d'une liste d'items *Marker*. Chacun d'entre eux peut être utilisé comme un élément de l'interface. On verra comment plus tard.

Voici à quoi ressemble le composant *MarkerPanel* :

```
// MarkerPanel.qml

import QtQuick 1.1

Rectangle {
    id: root
    width: 50
    height: 300
    // Élément Column qui gère le
    positionnement relatif.
    Column {
        id: layout
        anchors.fill: parent
        spacing: 10
        Repeater {
            // Trois items Marker.
            model: 3
            delegate: Marker { id:
marker }
        }
    }
}
```

Note : tester chaque composant individuellement pendant la phase de prototypage est toujours une bonne idée. Cela permet de repérer les erreurs dès que possible.

Si on lance le fichier *MarkerPanel.qml* avec *Qt Creator* et son *qmlviewer*, on devrait obtenir ceci :



3.4.1. Et ensuite ?

Dans la prochaine partie, on verra comment assembler les composants créés jusqu'à présent et on en finira avec le prototype.

3.5. Finalisation du prototype

On a enfin nos composants prêts à servir dans l'interface ! Voici la liste de ceux que nous avons implémentés pour l'instant :

- *Note* ;
- *NoteToolbar* ;
- *Marker* ;
- *MarkerPanel* ;
- *Page*.

On aura besoin de bien d'autres en avançant, comme vous

pourrez le constater.

Comme dit précédemment, Qt Creator génère automatiquement un fichier *main.qml* considéré comme le fichier principal lors du lancement de l'application. On assemble donc ces composants dans ce fichier.

3.5.1. Assembler le prototype

Si on regarde de nouveau le concept d'interface, on remarque que la zone des signets est située sur la droite. La zone qui contiendra les notes (*Page*) se situe au centre. Il manque encore la barre d'outils.

Implémentons-la donc maintenant !

La barre d'outils contient deux boutons : un pour créer une nouvelle note et l'autre pour les supprimer toutes. Pour rendre le tout plus simple, on ne les sépare pas dans un nouveau fichier QML, on les déclare à l'intérieur même du fichier *main.qml*.

Le code ressemble à ceci :

```
// Utilisation de l'élément Rectangle pour notre
barre d'outils.
// Il aide à aligner le Column avec le reste des
items.
Rectangle {
    id: toolbar

    // On définit une largeur puisque toolbar
ne sera
    // pas attaché sur la droite.
    width: 50

    color: "#444a4b"
    anchors {
        left: window.left
        top: window.top; bottom:
window.bottom
        topMargin: 100; bottomMargin: 100
    }

    // Utilisation de Column pour gérer nos
deux outils.
    Column {
        anchors { anchors.fill: parent;
topMargin: 30 }
        spacing: 20

        // Juste pour le prototype, on
utilise
        // un repeater pour générer deux
outils.
        Repeater {
            model: 2
            // Le rectangle
symbolisera l'outil
            // pendant la phase de
prototypage.
            Rectangle { width: 50;
height: 50; color: "red" }
        }
    }
}
```

Maintenant, nous sommes fins prêts à assembler le prototype. Voici le fichier *main.qml* :

```
// main.qml

import QtQuick 1.1

Rectangle {
    // Utilisation de window comme id.
    id: window
    width: 800
    height: 600

    // Création de MarkerPane.
    MarkerPanel {
        id: markerPanel
        width: 50
        anchors.topMargin: 20
        anchors {
            right: window.right
            top: window.top
            bottom: window.bottom
        }
    }

    // Barre d'outils.
    Rectangle {
        id: toolbar
        width: 50
        color: "#444a4b"
        anchors {
            left: window.left
            top: window.top
            bottom: window.bottom
            topMargin: 100
            bottomMargin: 100
        }
    }
    Column {
        anchors { fill: parent;
topMargin: 30 }
        spacing: 20
        Repeater {
            model: 2
            Rectangle { width: 50;
height: 50;
color: "red"
        }
    }
}
```

Voici à quoi ressemble le tout assemblé lorsqu'il est lancé avec *Qt Creator* :



3.5.2. Rendre les notes déplaçables avec l'élément QML `MouseArea`

Pour l'instant, on a un prototype très simpliste qui servira de base pour plus tard. On peut cependant ajouter une fonctionnalité très simple qui permettra de déplacer les notes sur la page dès maintenant. L'élément QML `MouseArea` possède un ensemble de propriétés `drag`. La propriété `drag.target` contiendra l'identificateur de notre note.

Comme l'utilisateur doit utiliser la barre d'outils des notes pour les déplacer, cet élément (le `MouseArea`) devra se trouver dans le composant `NoteToolBar`. Ce composant gère les déplacements avec la souris.

Pour réussir ce tour de force, on doit permettre à l'item `NoteToolBar` de lier sa propriété `drag.target` à l'id du composant `Note`. On utilise à cette fin les alias de propriétés (ce sont en fait des références à d'autres propriétés).

Voici donc le code de `NoteToolBar` modifié :

```
// NoteToolBar.qml
import QtQuick 1.1

Rectangle {
    id: root
    width: 100
    height: 62
    color: "#9e964a"

    // Alias de propriété " drag " de cet
    item sur la propriété drag de mouseArea.
    property alias drag: mousearea.drag

    // Création de l'item MouseArea.
    MouseArea {
        id: mousearea
        anchors.fill: parent
    }
}
```

Dans le code du dessus, on peut voir que l'alias de propriété `drag` a été lié à la propriété `drag` de `MouseArea`. On verra comment l'utiliser dans le composant `Note`.

```
// Note.qml
import QtQuick 1.1

Rectangle {
    id: root
    width: 200
    height: 200
    color: "#cabflb"

    // Création d'un item NoteToolBar
    // qui sera positionné en fonction de ses
    parents.
    NoteToolBar {
        id: toolbar
        height: 40
        anchors {
            top: root.top
            left: root.left
        }
    }
}
```

```
        right: root.right
    }

    // Utilisation de l'alias de
    propriété défini plus haut
    // pour affecter à drag.target
    notre item Note.
    drag.target: root
}

// Création du TextEdit (zone de saisie).
TextEdit {
    anchors {
        top: toolbar.bottom
        bottom: root.bottom
        right: root.right
        left: root.left
    }
    wrapMode: TextEdit.WrapAnywhere
}
}
```

Vous pouvez trouver des informations plus détaillées sur l'utilisation de propriétés liées à cette la page.

3.5.2.1. Et ensuite ?

On commence la « vraie interface ». On ajoute quelques fonctionnalités de base.

4. Implémentation de l'interface utilisateur et ajout de quelques fonctionnalités

Le prototype de base a permis de définir la base de l'interface et des fonctionnalités requises et d'identifier les composants QML requis.

En se basant sur le travail déjà accompli, on tente de construire une interface plus complète et « sophistiquée ». On ajoutera aussi quelques nouvelles fonctionnalités.

Ce chapitre décrit les étapes détaillées de l'utilisation de graphismes avec QML, quelques améliorations de l'interface et l'ajout de fonctionnalités plus avancées avec JavaScript. On approfondira quelques éléments QML et on en découvrira de nouveaux au fur et à mesure que la complexité du code augmentera et que les fonctionnalités viendront s'ajouter.

Voici la liste des points abordés pendant ce chapitre :

- gestion des éléments `Page` avec l'aide de `Repeater` et d'un nouveau composant : `PagePanel` ;
- graphismes plus avancés avec QML ;
- fonctions JavaScript à même le code pour ajouter des fonctionnalités plus complexes ;
- approfondissement des éléments QML que l'on a pu voir jusqu'à présent.

4.1. Création du composant `PagePanel` pour gérer les `Pages`

Avec le composant `Page`, on n'a créé qu'une seule et unique page. On l'a attachée à ses parents, comme quasiment tous les autres items avec la propriété `anchor`. Cependant, dans le cahier des charges, on devait disposer

de plusieurs pages, sélectionnables avec les signets *Marker*. On a vu comment l'élément *MarkerPanel* pouvait en aligner trois verticalement. On reprend la même logique dans l'implémentation de *PagePanel*.

4.1.1. Utilisation d'un élément QML Item comme élément de base

Avant d'aller plus loin, il est nécessaire que vous compreniez pourquoi utiliser l'élément *Rectangle* comme élément principal d'un composant devrait être évité dans la mesure du possible. Jusqu'à maintenant, on l'a utilisé parce qu'il fournissait rapidement un résultat valable (visuellement parlant) et c'est généralement ce qu'on attend d'un prototype.

Une fois, cependant, que le prototype est terminé, c'est mieux de remplacer le *Rectangle* par un *Item*, surtout si on veut personnaliser l'arrière-plan un peu plus qu'en changeant la couleur de fond (c'est d'ailleurs ce qu'on verra dans une prochaine partie).

Le composant *Page* ressemble désormais à ceci :

```
// Page.qml

import QtQuick 1.1
Item {
    id: root
    ...
}
```

Attention : à partir de maintenant, on considérera que l'élément de base de chacun des composants est un *Item*. Jetez donc un coup d'œil au code source de ce chapitre.

4.1.2. Utilisation des états (states)

À première vue, la différence majeure entre *PagePanel* et *MarkerPanel* est que les *Marker* sont toujours visibles, alors qu'on est supposé ne voir qu'une seule page à la fois dans avec *PagePanel*.

On pourrait obtenir le résultat attendu de plusieurs façons. Notamment, on pourrait utiliser une fonction JavaScript qui modifierait la visibilité de chaque *Page* en fonction du *Marker* pressé par l'utilisateur.

Avec *NoteApp*, on utilisera l'élément *State* à cette fin. Le composant *PagePanel* possédera trois états différents et chacun sera lié à une seule page. Ainsi, une seule d'entre elles sera visible à la fois.

D'abord, on doit un peu modifier le composant *Page* en affectant à la propriété *opacity* la valeur 0.0 par défaut. L'état ne fera en fait que modifier l'opacité de la page qui lui est liée.

```
// Page.qml

import QtQuick 1.1
Item {
    id: root
    opacity: 0.0
    ...
}
```

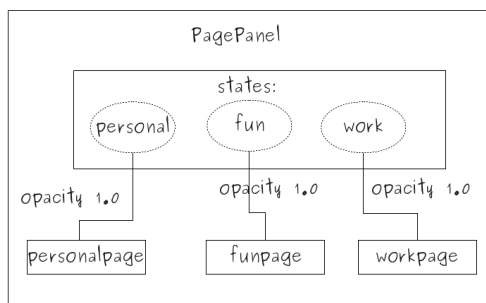
Une fois le fichier *PagePanel.qml* créé, on définit les différents états et leur page associée. On a besoin de ces états :

- personnel (*personal*) ;
- loisirs (*fun*) ;
- travail (*work*).

Ils vont modifier respectivement l'opacité de ces pages :

- *personalpage* ;
- *funpage* ;
- *workpage*.

Voici une représentation de ce qui a été décrit plus haut :



Voici à quoi ressemble le code de *PagePanel* :

```
import QtQuick 1.1

// PagePanel.qml

Item {
    id: root
    // Création de la liste d'états (states).
    states: [
        // Création d'un item state avec son nom.
        State {
            name: "personal"
            // Les propriétés qui vont être modifiées.
            PropertyChanges {
                target: personalpage
                opacity:1.0
                restoreEntryValues: true
            }
        },
        State {
            name: "fun"
            PropertyChanges {
                target: funpage
                opacity:1.0
                restoreEntryValues: true
            }
        },
        State {
            name: "work"
            PropertyChanges {
                target: workpage
                opacity:1.0
                restoreEntryValues: true
            }
        }
    ]

    // Création des trois pages.
    Page { id: personalpage; anchors.fill: parent }
}
```

```

Page { id: funpage; anchors.fill: parent }
Page { id: workpage; anchors.fill: parent }
}

```

Note : en mettant la propriété *restoreEntryValue* à *true*, on précise que, quand on change d'état, on veut que la propriété soit remise à sa valeur par défaut.

En regardant le code du dessus, on peut voir que l'on a créé trois items *Page* et que les trois états associés modifient l'opacité. Aucune surprise, donc.

Dans cette étape, on a réussi à créer le composant *PagePanel* qui permettra à l'utilisateur de changer de page et donc de mieux gérer ses notes.

4.1.2.1. Et ensuite ?

Dans la prochaine étape, les items *Marker* pourront changer l'état de *PagePanel*.

4.2. Lier les items *Marker* avec leur page respective dans le *Repeater*

On vient tout juste de voir l'implémentation du composant *PagePanel*, qui en fait utilise trois états différents pour modifier chacun l'opacité d'un composant *Page*. Dans cette étape, on verra comment utiliser les items *Marker* et *MarkerPanel* pour permettre à l'utilisateur de changer de page.

Pendant la phase de prototypage, le composant *MarkerPanel* était très simple et ne possédait aucune fonctionnalité. Il utilisait un élément *Repeater* qui générerait trois items *Marker* (on le lui avait précisé avec les délégués).

MarkerPanel devrait s'occuper de stocker le signet actif, celui cliqué par l'utilisateur. En se basant sur ce fait, *PagePanel* mettrait à jour son état courant. On doit donc lier l'état de *PagePanel* à une nouvelle propriété de *MarkerPanel* qui contiendrait le signet actif.

On ajoute donc une propriété *activeMarker* (de type *string*) :

```

// MarkerPanel.qml

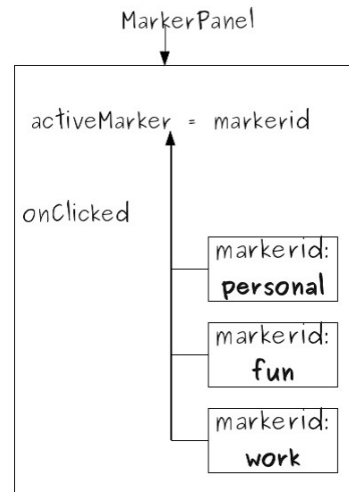
import QtQuick 1.1

Item {
    id: root
    width: 150; height: 450

    // Une propriété activeMarker de type
    string pour stocker le signet courant.
    property string activeMarker: "personal"
    ...
}

```

On pourrait stocker une valeur *markerid*, utile pour identifier de façon unique chaque signet. De cette manière, *activeMarker* prendrait alors la valeur *markerid* du signet activé par l'utilisateur.



L'élément *Repeater* génère les trois signets d'après un modèle. On peut donc modifier les valeurs de la propriété *markerid*.

```

// MarkerPanel.qml

import QtQuick 1.1

Item {
    id: root
    width: 150; height: 450

    // Une propriété activeMarker de type string
    // pour stocker le signet courant.
    property string activeMarker: "personal"

    // Une liste de données pour les trois signets
    // (le modèle).
    property variant markerData: [
        { markerid: "personal" },
        { markerid: "fun" },
        { markerid: "work" }
    ]

    Column {
        id: layout
        anchors.fill: parent
        spacing: 5

        Repeater {
            // Application du modèle
            model: markerData
            delegate:
                Marker {
                    id: marker
                    // Gestion du signal " clicked() " de
                    // l'item Marker. Affecte la valeur markerid
                    // du signet cliqué à la propriété
                    // activeMarker de MarkerPanel.
                    onClicked: root.activeMarker =
                    modelData.markerid
                }
        }
    }
}

```

Dans ce code, on modifie la propriété *activeMarker* quand l'événement *onClicked* est actionné. Ceci veut dire que le signal *clicked()* est préalablement défini dans le composant *Marker* ; il est utilisé si l'utilisateur clique sur le signet.

Voici donc le composant *Marker* modifié :

```
// Marker.qml

Item {
    id: root
    width: 50; height: 90
    signal clicked()

    MouseArea {
        id: mouseArea
        anchors.fill: parent
        // On émet le signal "clicked()".
        onClicked: root.clicked()
    }
}
```

Pour l'instant, le composant *PagePanel* est capable de changer de page (en réalité, en mettant l'opacité à 0 ou à 1). *MarkerPanel* est lui capable d'identifier le signet courant et d'envoyer un signal quand il change.

Comment utiliser la propriété *activeMarker* de *MarkerPanel* pour mettre jour l'état de *PagePanel* ?

Dans le fichier *main.qml*, où on a déjà un item *MarkerPanel* et une page, on doit d'abord remplacer l'item *Page* avec *PagePanel*.

```
// main.qml

// Création d'un MarkerPanel.

MarkerPanel {
    id: markerPanel
    width: 50
    anchors.topMargin: 20
    anchors {
        right: window.right
        top: window.top
        bottom: window.bottom
    }
}

...

// Création d'un PagePanel.
PagePanel {
    id: pagePanel
    // On lie l'état de PagePanel à la
    propriété
    // activeMarker de MarkerPanel.
    state: markerPanel.activeMarker
    anchors {
        right: markerPanel.left
        left: toolbar.right
        top: parent.top
        bottom: parent.bottom
        leftMargin: 1
        rightMargin: -50
        topMargin: 3
        bottomMargin: 15
    }
}
```

Dans le code du dessus, on peut voir comment le *property*

binding peut aider à lier la propriété *state* à *activeMarker*. Cela veut dire que, dès que *activeMarker* sera modifié, *state* réagira en conséquence et sera modifié, changeant ainsi la page courante.

4.2.1. Et ensuite ?

On modifiera un peu l'application pour la rendre plus attrayante, à l'aide de graphismes.

4.3. Ajouter des graphismes : Image ou BorderImage?

Grâce à la nature de QML, les développeurs et les designers peuvent travailler ensemble tout le long de la phase de développement. De nos jours, avoir une interface propre et classe fait une grosse différence quant à la façon dont l'utilisateur perçoit l'application.

N'hésitez surtout pas à utiliser autant de graphismes que vous le souhaitez dans votre interface. La collaboration entre les développeurs et les graphistes est très efficace avec QML, puisque ces derniers peuvent directement tester leurs graphismes sur des éléments de l'interface, mais aussi comprendre d'un point de vue technique ce dont les développeurs ont réellement besoin. L'application n'en devient que plus attrayante et plus facile à maintenir.

Ajoutons donc des graphismes à nos composants.

4.3.1. Images d'arrière-plan sur les composants QML

L'élément *BorderImage* est conseillé dans les cas où la taille de l'image doit être ajustée en fonction du contexte, tout en conservant les bords à leur taille initiale. Les effets d'ombre sur les composants sont un bon exemple : en général, on préférerait qu'elles conservent la même taille.

Regardons ensemble comment utiliser les *BorderImage* comme arrière-plan pour les composants.

On ajoute l'élément *BorderImage* dans *PagePanel.qml*.

```
// PagePanel.qml

...

BorderImage {
    id: background
    // BorderImage prend toute la place
    possible.
    anchors.fill: parent
    source: "images/page.png"
    // On indique la marge.
    // Cette information devrait venir du
    graphiste.
    border.left: 68; border.top: 69
    border.right: 40; border.bottom: 80
}

...
```

On fait la même chose à l'intérieur de *Note* dans *Note.qml*.

```
// Note.qml

...

BorderImage {
```

```

id: noteImage
anchors { fill: parent}
source: "images/personal_note.png"
border.left: 20; border.top: 20
border.right: 20; border.bottom: 20
}

// Création de NoteToolbar qui sera positionné en
fonction de son parent.
NoteToolbar {
    id: toolbar
    ...
}

```

Attention : soyez sûr que l'élément *BorderImage* soit utilisé au bon « endroit », parce que l'ordre dans lequel sont définis les éléments d'un composant affecte le rendu final. Les items avec la même valeur « Z » (la profondeur) apparaîtront ainsi dans l'ordre dans lequel ils sont déclarés. Référez-vous à la page de la documentation pour plus d'informations à ce sujet : [Lien 48](#).

Vous vous demandez sûrement quelle est la meilleure approche pour utiliser un arrière-plan sur les items *Marker* alors qu'ils sont créés à l'intérieur du composant *MarkerPanel*. Eh bien, la solution se trouve déjà dans le code du composant *MarkerPanel* !

La liste *markerData*, le modèle du *Repeater*, existe déjà. On peut agrandir *markerData* pour qu'elle contienne les chemins relatifs des images et utiliser *Image* comme élément de base de *Marker*.

```

// Marker.qml

import QtQuick 1.1

// Image est très pratique comme item de base
puisque Marker est simplement
// un élément graphique avec un signal clicked().
Image {
    id: root

    // Déclaration du signal " clicked() ".
    signal clicked()

    // Création de MouseArea pour détecter
les clics de souris.
    MouseArea {
        id: mouseArea
        anchors.fill: parent

        // On émet le signal " clicked()
".
        onClicked: root.clicked()
    }
}

```

Voyons maintenant comment le composant *MarkerPanel* doit être modifié :

```

// MarkerPanel.qml
...
// Dans markerData, on ajoute les chemins des
images à utiliser.
property variant markerData: [
{ img: "images/personalmarker.png", markerid:
"personal" },
{ img: "images/funmarker.png", markerid: "fun" },

```

```

{ img: "images/workmarker.png", markerid:
"work" }
]

Column {
    id: layout
    anchors.fill: parent
    spacing: 5

    Repeater {
        // On applique le modèle.
        model: markerData
        delegate: Marker {
            id: marker

            // On lie la valeur img du modèle
// à la valeur source du signet.
            source: modelData.img

            onClicked: root.activeMarker =
modelData.markerid
        }
    }
}

```

Dans ce code, on peut voir comment la propriété source de *Marker* est liée à la valeur *img* de *markerData*.

On utilise aussi l'élément *BorderImage* pour mettre un arrière-plan sur le composant *NoteToolbar* ainsi que pour l'élément principal dans *main.qml*.

Note : demandez toujours aux graphistes quelles devraient être les marges sur les contours des éléments *BorderImage* et comment ces éléments devraient être alignés et positionnés.

En lançant *MarkerPanel.qml* dans Qt Creator, vous devriez obtenir ceci :



4.3.2. Création du composant Tool

Ce serait plus pratique de créer un unique composant qui pourrait être utilisé pour les boutons *New Note* (nouvelle note) et *Clear All* (tout supprimer). C'est pourquoi on implémente à présent un composant *Tool*, qui aura comme élément de base *Image* qui gèrera les clics de souris de l'utilisateur.

L'élément *Image* est souvent utilisé comme élément de l'interface de base, quelle que soit sa nature (image statique ou animée).

```

// Tool.qml

```

```

import QtQuick 1.1

// Utilisation de Image comme élément de base.
Image {
    id: root

    // On ajoute le signal clicked().
    signal clicked()

    // Utilisation de MouseArea
    // pour intercepter les clics de souris.
    MouseArea {
        anchors.fill: parent
        // Émission du signal clicked.
        onClicked: root.clicked()
    }
}

```

On utilise le composant *Tool* pour créer une barre d'outils, puisqu'il est déjà disponible. On doit modifier le code du prototype pour utiliser des items *Tool* au lieu de simples rectangles.

```

// main.qml
...
// Arrière plan de la barre d'outils
Rectangle {
    anchors.fill: toolbar
    color: "white"
    opacity: 0.15
    radius: 16
    border { color: "#600"; width: 4 }
}

// Utilisation de Column pour aligner
// les Tool horizontalement.
Column { // Barre d'outils de gauche
    id: toolbar
    spacing: 16
    anchors {
        top: window.top
        left: window.left
    }
}

```

```

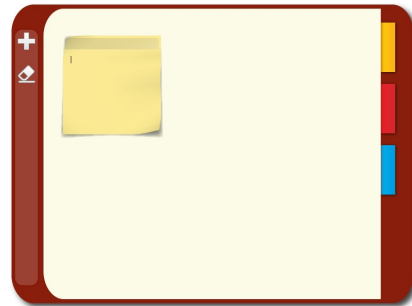
        bottom: window.bottom
        topMargin: 50
        bottomMargin: 50
        leftMargin: 8
    }

    // Outil " nouvelle note "
    Tool {
        id: newNoteTool
        source: "images/add.png"
    }

    // Outil " tout supprimer "
    Tool {
        id: clearAllTool
        source: "images/clear.png"
    }
}
...

```

Maintenant que tous les graphismes sont en place, l'application est normalement beaucoup plus attrayante.



4.3.2.1. Et ensuite ?

Dans le prochain chapitre, on regardera en détail comment créer et gérer les notes dynamiquement et comment les stocker localement.

Retrouvez l'article de Nokia traduit par Grégoire Lothe en ligne : [Lien 49](#)

Comment rechercher une valeur dans une table qui contient des paliers

Il s'agit de rechercher dans une table, une valeur qui ne s'y trouve pas nécessairement et de choisir selon les circonstances : la valeur immédiatement supérieure (ou éventuellement égale) ou la valeur immédiatement inférieure.

Access offre plusieurs voies pour atteindre ce but.

Dans ce tutoriel, nous utiliserons uniquement des fonctions intégrées sans recourir à du code VBA.

Nous aborderons l'utilisation des fonctions intégrées au moyen de quelques exemples pour illustrer la recherche d'une date, d'une heure, d'un texte ou d'une valeur numérique dans une table.

1. Ce dont il s'agit

Des dessins plutôt qu'un discours :

DateChangement	Taux
6/11/08	3,250
4/12/08	2,500
15/01/09	2,000
2/04/09	1,250
7/05/09	1,000
7/04/11	1,250
7/07/11	1,500
3/11/11	1,250
8/12/11	1,000

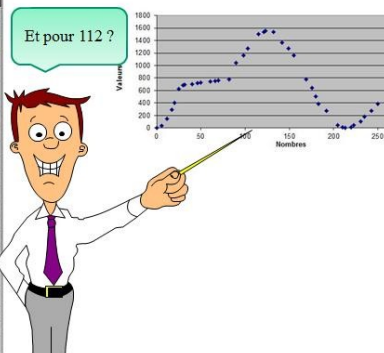


HeureDepart
06.37
07.13
07.37
08.15
08.37
09.13
10.15
11.15
12.37
14.37
16.15
17.15
17.43
18.13
18.37



Signe	Debut
Verseau	0121
Poissons	0219
Bélier	0321
Taureau	0420
Gémeaux	0521
Cancer	0622
Lion	0723
Vierge	0823
Balance	0923
Scorpion	1023
Sagittaire	1123
Capricorne	1222

Nombre	Valeur
70	765
82	775
90	1040,055611
98	1162,5
103	1273,160398
115	1503,261781
121	1538,226009
123	1550
132	1538,226009
142	1368,684443
149	1273,160398
155	1162,5
169	775
176	640,4226623
180	509,9343889
183	387,5
192	276,8396025
205	46,73821889
210	11,77399142
213	0
220	11,77399142
223	46,73821889
231	103,8303121
232	46,73821889



Si la valeur n'est pas dans la table, ce qui nous intéresse pour la prise de décision, c'est de connaître les deux bornes qui encadrent la valeur recherchée. Selon le contexte, on choisira alors :

- soit la borne inférieure, par exemple pour un tarif valable à partir de telle date ;
- soit la borne supérieure, par exemple pour le départ du prochain train ;
- voire une extrapolation basée sur ces deux valeurs...

2. Prérequis

Savoir consulter l'aide d'Access.



D'une manière générale, pour se documenter sur les propriétés d'un formulaire ou d'un état, ou de leurs contrôles :

- afficher l'objet en mode création ;
- cliquer sur la propriété, elle se met alors en surbrillance ;

— enfoncer la touche <F1>.

L'aide Access s'ouvre à la bonne page.

On peut aussi :

— ouvrir l'aide <F1>, choisir l'onglet « Aide intuitive » et suivre les instructions ;

— ouvrir la fenêtre d'exécution (<Ctrl> + G), saisir un mot-clé, y placer le curseur de la souris et presser <F1>.

3. L'objectif pédagogique

Pour chaque exemple concret, nous allons construire pas à pas un formulaire dans lequel l'utilisateur saisit une valeur et reçoit la réponse à sa question. L'objectif est de proposer, en détail, **une** démarche intellectuelle pour aborder le problème et le résoudre sans faire appel à du code VBA.

3.1. Liste des fonctions utilisées dans les exemples

Date()

Format()

MaxDom()

MinDom()

Nz()

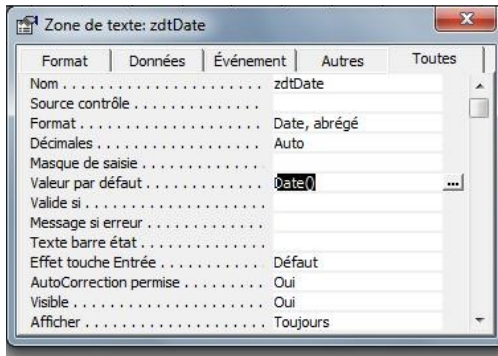
RechDom()

Replace()

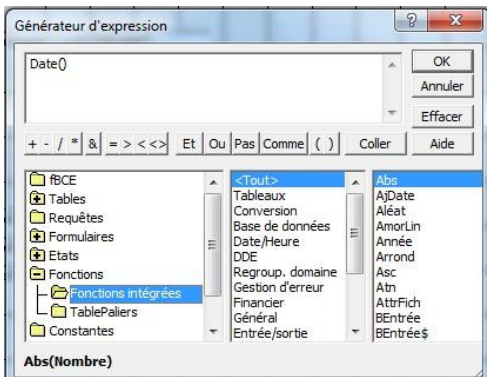
VraiFaux()

Si vous voulez une idée des principales fonctions intégrées :

- cliquez sur la propriété *Valeur par défaut* et ensuite sur les trois points qui apparaissent à droite.

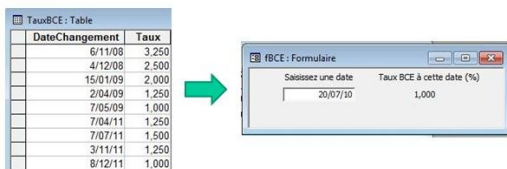


et choisissez :

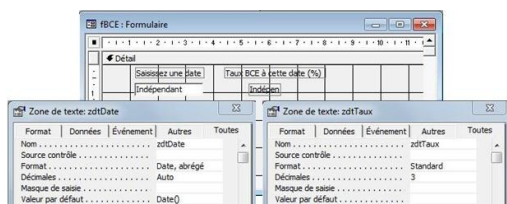


4. Taux directeur en vigueur à la BCE à telle date

Dans ce chapitre, nous recherchons une valeur de format *Date* dans une table.



Créons un formulaire *fBCE* avec deux contrôles : *zdtDate* et *ZdtTaux* :



Ce formulaire est indépendant : il n'est relié à aucune source.

Remarquez la valeur par défaut de *zdtDate* : *Date()*, c'est une fonction intégrée d'Access qui affiche la date du jour. C'est donc la date d'aujourd'hui qui sera affichée — par défaut — à l'ouverture du formulaire. L'utilisateur saisira la date de son choix.

4.1. Les étapes du raisonnement

4.1.1. La question est-elle pertinente

La table renseigne des taux à partir du 6/11/08, dans notre formulaire, cela n'a pas de sens de demander le taux d'une date antérieure au 6/11/08.

Convenons que si la date introduite dans *zdtDate* est antérieure au 6/11/08, nous afficherons le texte : « Date invalide » dans le contrôle *zdtTaux*.

La fonction intégrée *VraiFaux()* convient pour ce genre de traitement, sa syntaxe est :

```
VraiFaux («Expr»;«SiVrai»;«SiFaux»)
```

Nous devons remplacer «Expr» par la traduction en « Access » du français « la date introduite en *zdtDate* est antérieure au 6/11/08 ».

Nous devons remplacer «SiVrai» par le texte : « Date invalide ».

Nous devons remplacer «SiFaux» par le taux en vigueur à la BCE à la date mentionnée. À ce stade du raisonnement, contentons-nous d'afficher le texte « On verra plus tard ».

Voici la syntaxe de la source de *zdtTaux* :

```
=VraiFaux ([zdtDate]<#6/11/08#;"Date invalide";"On verra plus tard")
```

Remarquez :

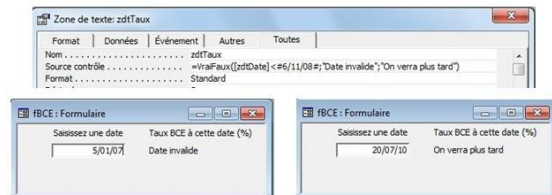
le signe « = » devant la fonction dans la syntaxe de la propriété *Source* du contrôle ;

les crochets « [] » qui encadrent le nom du contrôle qui contient la valeur à comparer ;

les croisillons « # » qui encadrent la date ;

les guillemets doubles droits « " » qui encadrent les chaînes de caractères.

Testons le résultat :



Généralisons, au lieu de demander si *zdtDate* est antérieure au 6/11/08, demandons plutôt si *zdtDate* est antérieure à la plus petite *DateChangement* contenue dans la table *TauxBCE*.

La fonction intégrée *MinDom()* permet de trouver cette valeur :

```
MinDom («Expr»;«Domaine»;«Critère»)
```

«Expr» est le nom du champ à rechercher : en l'occurrence « *DateChangement* ».

«Domaine» est l'espace dans lequel rechercher : la table « *TauxBCE* ».

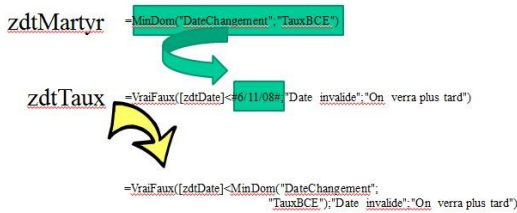
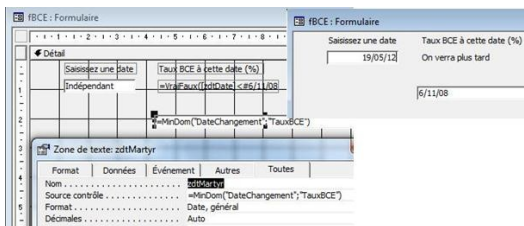
«Critère» permettrait de spécifier une condition pour préciser le choix. Cela n'est pas d'application dans le cas ici où nous demandons la plus petite date, sans restriction.

Ce qui nous donne :

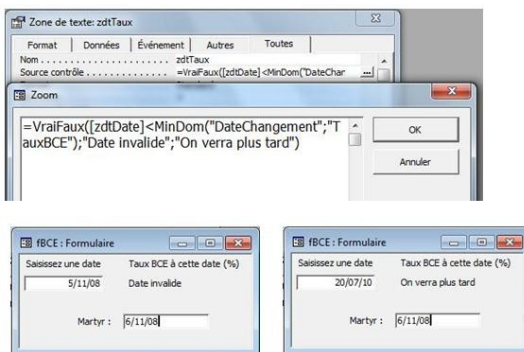
```
=MinDom("DateChangement";"TauxBCE")
```

Pour le vérifier, nous allons ajouter — provisoirement — un contrôle *zdtMartyr* dans notre formulaire. Cela nous permettra de progresser en vérifiant pas à pas que la voie est bonne.

Dans la source de *zdtTaux*, nous allons remplacer #6/11/08# par son expression plus générale :



À ce stade de notre progression, le formulaire fonctionne correctement, si ce n'est que le taux n'est pas affiché lorsque la date est valide.



Plus tard, c'est maintenant.

4.2. Comment trouver le taux à cette date ?

On va procéder en deux temps.

On va d'abord chercher la date du dernier changement par rapport à la date saisie dans *zdtDate*.

Quand on aura cette date, on recherchera le taux qui a alors été mis en vigueur.

4.2.1. Quelle est la date du dernier changement de taux ?

C'est, dans *TauxBCE*, la plus grande date qui est antérieure — ou éventuellement égale — à la date saisie en *zdtDate*.

C'est l'occasion de faire connaissance avec la fonction intégrée :

```
MaxDom («Expr»;«Domaine»;«Critère»)
```

«Expr» : *DateChangement*.

«Domaine» : *TauxBCE*.

«Critère» : il faudra exprimer « qui est inférieure ou égale au contenu de *zdtDate* ».

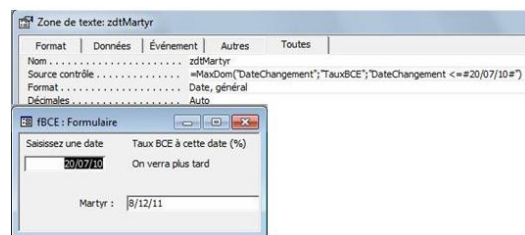
Progressons pas à pas.

Tentons ceci dans *zdtMartyr* :

```
=MaxDom("DateChangement";"TauxBCE";"DateChangement <=#20/07/10#")
```

On s'attend à trouver : le 7 mai 2009, c'est-à-dire la date du dernier changement de taux avant le 20 juillet 2010.

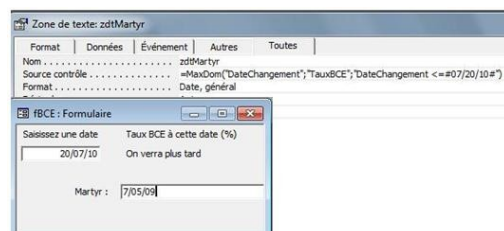
Et pourtant :



Pourquoi ? Parce que Access a interprété #20/07/10# comme étant le 10 juillet 2020 !

Dans cette fonction, Access attend la date présentée à l'anglo-saxonne mm/jj/aa, en l'occurrence #07/20/10#.

Voici :



Dans les pays francophones, le format d'une date est généralement jj/mm/aa.

Dans les pays anglophones c'est mm/jj/aa.

Et par exemple au Japon aa/mm/jj ou plutôt : 火曜日年月日.

Dans Access, cela dépend du contexte, parfois il attend #jj/mm/aa# par exemple dans la fonction *VraiFaux()*, parfois il attend #mm/jj/aa#, comme ici, dans *MaxDom()*.

L'interprétation par Access est parfois déconcertante :

Valeur	Interprétation
#01/07/10#	« 7 janvier 2010 »
#02/07/10#	« 7 février 2010 »
#03/07/10#	« 7 mars 2010 »
[...]	
#12/07/10#	« 7 décembre 2010 »
#13/07/10#	« 10 juillet 2013 »
#14/07/10#	« 10 juillet 2014 »
#2012/7/20#	« 20 juillet 2012 »
#20/7/2012#	« 20 juillet 2012 »

Continuons notre parcours du combattant, au lieu de nous référer à une date codée en dur, nous allons exprimer qu'il faut se référer au contenu de *zdtDate*.

Dans la source actuelle de *zdtMartyr* :

```
=MaxDom("DateChangement";"TauxBCE";"DateChangement <=#07/20/10#")
```

nous allons, dans le critère de la fonction, remplacer « 07/20/10 » par « ce qui se trouve en *zdtDate* sous la forme mm/jj/aa ».

Quelque chose comme :

le texte : "DateChangement <=#" ;
concaténé avec la date bien formatée de *zdtDate* ;
concaténé avec le texte : "#".

Pour bien formater le contenu de *zdtDate*, la syntaxe est :

```
Format([zdtDate];"mm/dd/yy")
```

Remarquez l'équivalent anglo-saxon "mm/dd/yy" de "mm/jj/aa". (OK, tout serait plus simple si la planète entière parlait français ! Mais il y a eu cette tour à Babel...)

Concaténer se traduit par l'esperluette « & ».

La source de *zdtMartyr* devient :

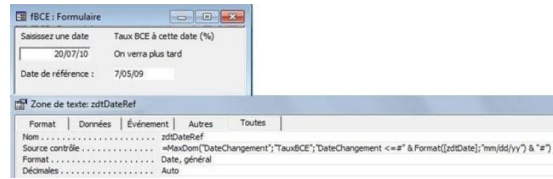
```
=MaxDom("DateChangement";"TauxBCE";"DateChangement <=#" & Format([zdtDate];"mm/dd/yy") & "#")
```

C'est la date du dernier changement de taux avant la date saisie par l'utilisateur. C'est elle qui va nous servir pour trouver le taux alors en vigueur.

Pour plus de clarté, rebaptisons le contrôle *zdtMartyr* en

zdtDateRef.

Voici où nous en sommes actuellement :



4.2.2. Cherchons maintenant le taux qui a été alors mis en vigueur à la BCE

La fonction intégrée *RechDom* convient dans ce cas :

```
RechDom («Expr»;«Domaine»;«Critère»)
```

«Expr» : le nom de la valeur cherchée dans le domaine, en l'occurrence *Taux* ;

«Domaine» : le nom du domaine. Ici, la table *TauxBCE* ;

«Critère» : nous allons exprimer ici qu'il faut choisir l'enregistrement qui a pour *DateChangement* celle que nous avons logée dans *zdtDateRef*.

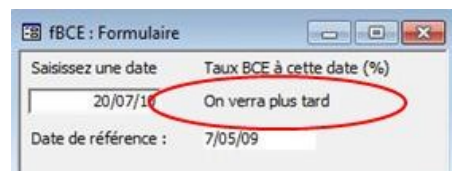
Ici aussi, le contexte veut que l'on reformate la date.

La syntaxe pour trouver le taux est donc :

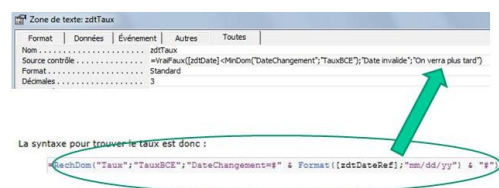
```
=RechDom("Taux";"TauxBCE";"DateChangement=#" & Format([zdtDateRef];"mm/dd/yy") & "#")
```

4.3. On est au bout

La dernière retouche :

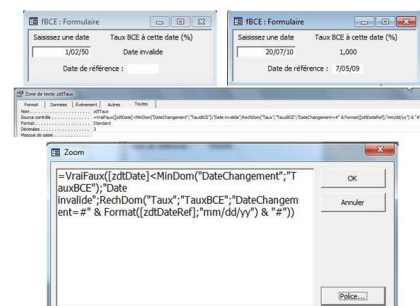


C'est tout vu !



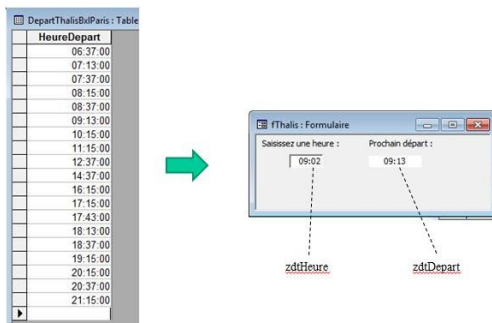
N.B. Sans le « = » initial.

On y est :



5. Le prochain Thalys

Dans ce chapitre, nous recherchons une valeur de format *Heure* dans une table.



5.1. Les étapes du raisonnement

5.1.1. Le cas banal

Avec l'heure saisie, il faut trouver dans la table *DepartThalisBxlParis* la valeur de *HeureDepart* immédiatement supérieure à l'heure saisie dans *zdtHeure*.

Si on s'inspire de notre explication précédente, la fonction intégrée :

```
MinDom («Expr»;«Domaine»;«Critère»)
```

répondra à cette demande.

Nous aurons :

«Expr» : *HeureDepart* ;

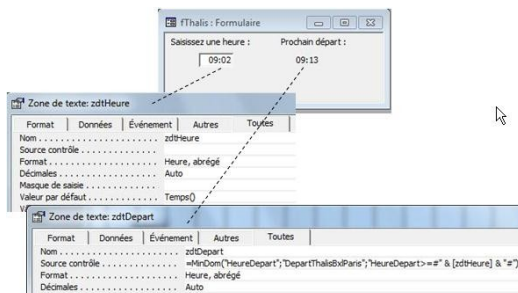
«Domaine» : *DepartThalisBxlParis* ;

«Critère» : l'heure qui est supérieure à celle mentionnée dans *zdtHeure*.

```
=MinDom("HeureDepart";"DepartThalisBxlParis";"HeureDepart">=#" & [zdtHeure] & "#")
```

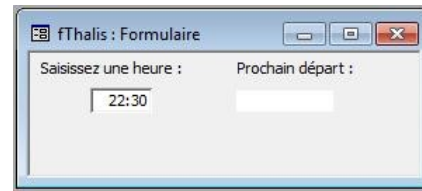
Remarquez l'analogie avec la syntaxe que nous avons adoptée pour une date : l'expression de l'heure est encadrée de croisillons « # ».

Une différence toutefois : il n'est pas nécessaire de personnaliser le format. Quel que soit le contexte, c'est partout « hh:mm[:ss] ».



5.1.2. Cas particulier : l'heure saisie est plus tard que le dernier train

Dans l'état actuel, voici ce que cela donne :



Rien ne s'affiche : en fait, puisque dans *DepartThalisBxlParis*, il n'y a pas d'*HeureDepart* qui est supérieure à 22:30.

Deux conséquences :

- la première, le résultat de :

```
=MinDom("HeureDepart";"DepartThalisBxlParis";"HeureDepart">=#" & [zdtHeure] & "#")
```

est la valeur Null ;

- la seconde, vous passerez la nuit à Bruxelles et vous prendrez le train suivant, c'est-à-dire la plus petite valeur *HeureDepart* dans la table *DepartThalisBxlParis*.

C'est l'occasion d'utiliser la fonction intégrée :

```
Nz («Expr»;«ValeurSiNull»)
```

Avec cette fonction, on peut présenter une alternative :

avec «Expr», on exprime ce qu'il faut faire dans le cas banal ;

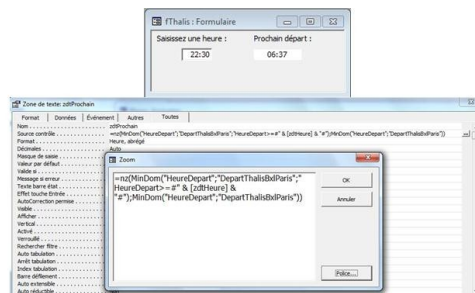
avec «ValeurSiNull», on exprime ce qu'il faut faire si la première branche donne un résultat égal à Null.

Dans notre cas, la deuxième branche de l'alternative sera le premier train :

```
MinDom("HeureDepart";"DepartThalisBxlParis")
```

sans paramètre pour le critère.

Et notre formulaire est opérationnel :



Bon voyage !

6. Quel signe zodiacal ?

Signe	Debut
Verseau	0121
Poissons	0219
Bélier	0321
Taureau	0420
Gémeaux	0521
Cancer	0622
Lion	0723
Vierge	0823
Balance	0923
Scorpion	1023
Sagittaire	1123
Capricorne	1222



Dans ce chapitre, nous recherchons une valeur de format *Texte* dans une table.

Les quatre chiffres de *Debut* (en format *Texte*) représentent, sous la forme mmjj, la date de début du *Signe* mentionné.

6.1. Le cas banal

Il faut choisir *Signe* dans l'enregistrement de la table *SignesZodiaque* dont *Debut* est immédiatement inférieur ou égal à la date saisie, présentée sous la forme mmjj.

Exemple : pour 28 janvier 1955, il faut prendre « **Verseau** », car « **0121** » est la valeur immédiatement inférieure à « **0128** ».

On va donc opérer en deux temps :

- trouver la plus grande valeur de *Debut* qui est inférieure ou égale au mmjj de la date saisie dans *zdtNaissance* ;
- avec ce *Debut*, chercher dans *SignesZodiaque* le *Signe* correspondant.

Déjà vu !

```
MaxDom («Expr»;«Domaine»;«Critère»)
```

«Expr» : *Debut*.

«Domaine» : *SignesZodiaque*.

«Critère» :
le texte « *Debut* <= » concaténé avec la valeur de `Format([zdtNaissance];"mmdd")` encadrée d'une paire de doubles-quotes (puisque c'est du texte).

«Critère» ressemblerait à ceci :

```
Debut <="0128"
```

le tout inclus dans une paire de **quotes**, il viendrait

```
"Debut <="0128"
```

Nous avons donc le signe « » qui est lui-même inclus dans une paire de « " ». La règle est alors de doubler les « " » intérieurs, ainsi :

```
"Debut <=""0128"""
```

Notre fonction s'écrira donc :

```
MaxDom("Debut";"SignesZodiaque";"Debut <="" & Format([zdtNaissance];"mmdd") & """)"
```

Comme nous l'avions fait plus haut, ajoutons temporairement dans notre formulaire un contrôle *zdtMartyr* et affectons-lui comme propriété *Source* :

```
=MaxDom("Debut";"SignesZodiaque";"Debut <="" & Format([zdtNaissance];"mmdd") & """)"
```

Il vient :

Recherchons maintenant le *Signe* qui correspond à cette valeur, d'abord en utilisant la valeur stockée dans *zdtMartyr* :

```
=RechDom("Signe";"SignesZodiaque";"Debut="" & [zdtMartyr] & """)"
```

que nous logeons dans la source de *zdtSigne*.

Vérifions le résultat :

C'est correct, même si, connaissant quelqu'un qui est né à cette date, on s'attendait plutôt à « s'agiter » comme réponse...

Et enfin, dans cette syntaxe remplaçons « *[zdtSigne]* » par sa formule, il vient :

```
=RechDom("Signe";"SignesZodiaque";"Debut="" & MaxDom("Debut";"SignesZodiaque";"Debut <="" & Format([zdtNaissance];"mmdd") & """) & """)"
```

... et nous pouvons supprimer *zdtMartyr*.

Et hop, ça marche.

On est presque au bout.

6.2. Et s'il n'y a pas de borne inférieure ?

Par exemple si la date de naissance est antérieure au 21 janvier, alors :

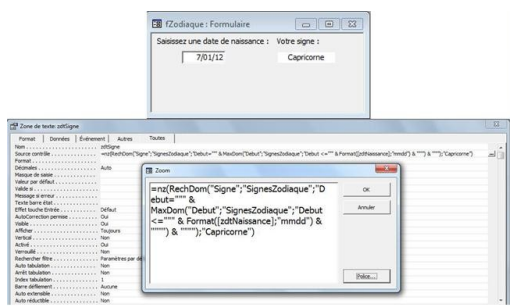
```
=RechDom("Signe";"SignesZodiaque";"Debut="" &
MaxDom("Debut";"SignesZodiaque";"Debut <="" &
Format([zdtNaissance];"mdd") & "")) & ""))
```

donnera un résultat de valeur Null, alors qu'il faudrait « Capricorne ».

Qu'à cela ne tienne, nous connaissons déjà la fonction Nz().

Nous aménageons une dernière fois, la source de *zdtSigne*, qui devient :

```
=nz(RechDom("Signe";"SignesZodiaque";"Debut="" &
MaxDom("Debut";"SignesZodiaque";"Debut <="" &
Format([zdtNaissance];"mdd") & "")) &
"";"Capricorne")
```

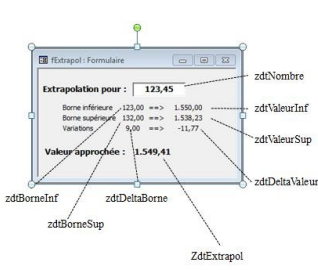


Et nous y sommes !

7. Extrapolation entre deux bornes

Dans ce chapitre, nous recherchons une valeur numérique dans une table.

Nombre	Valeur
0	0
2,5	6,25
12	144
17	289
20	400
25	625
30	895
32	899
40	705
46	717
50	725
61	747
66	757
70	765
82	775
90	1040,065611
98	1162,5
103,5	1273,160398
115	1503,261781
121	1538,226009
123	1550
132	1538,226009
142	1368,684443
149	1273,160398
155	1162,5
169	775
176	640,4226923
180	509,9343889
183	387,5
192	276,8396025
205	46,73821899
210	11,77399142
213	0
220	11,77399142
223	46,73821899
231	103,8303121
235	181,3155566
243	276,8396025
250	387,5

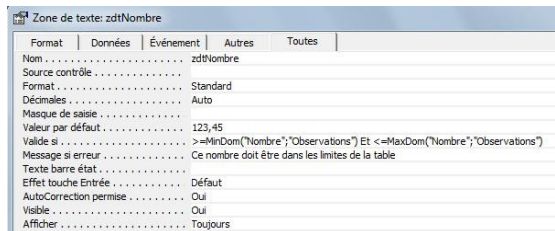


7.1. La demande est-elle fondée ?

Commençons par exprimer que le nombre entré en *zdtNombre* est dans les limites de la table, c'est-à-dire : supérieur ou égal au plus petit *Nombre* de la table et inférieur ou égal au plus grand.

```
>=MinDom("Nombre";"Observations") Et
<=MaxDom("Nombre";"Observations")
```

Utilisons les propriétés *Valide si* et *Message si erreur* du contrôle *zdtNombre* pour installer cette fonctionnalité dans le formulaire :



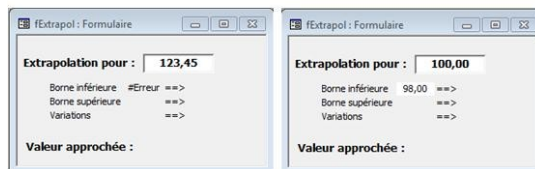
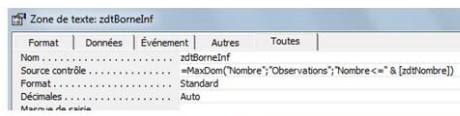
7.2. Recherche de la borne inférieure

Pour la source de *zdtBorneInf*, on s'attend à une syntaxe comme celle-ci :

```
=MaxDom("Nombre";"Observations";"Nombre<=" &
[zdtNombre])
```

Remarquez que [zdtNombre] n'est pas encadré de délimiteurs - pas de doubles-quotes ni de croisillons - puisqu'il s'agit d'une valeur numérique !

Testons :



Cela ne fonctionne pas lorsqu'on cherche la borne inférieure avec 123,45. Par contre, on la trouve avec 100.

Access s'attend à un point « . » comme symbole décimal. Dans vos paramètres régionaux, vous avez probablement choisi la virgule « , » comme symbole pour séparer la partie entière et les décimaux. En fait, il faut chercher avec « 123.45 » et non avec « 123,45 ».

Il nous faut donc remplacer la virgule par un point dans le contenu de *zdtNombre*.

La fonction *Replace* répond à ce but.

La syntaxe est la suivante :

```
Replace(«la chaîne de caractères originale»;«il y a»;«il faut»)
```

Pour plus de détails sur cette fonction, consultez l'aide (<F1> et saisissez « replace fonction » dans l'onglet *Aide Intuitive* puis <Entrée>).

En l'occurrence, la source de *zdtBorneInf* devient :

```
=MinDom("Nombre";"Observations";"Nombre>=" &
Replace([zdtNombre];",",";"))
```

7.3. Recherche de la valeur inférieure

Il suffit de rechercher dans la table *Observations* la *Valeur* qui correspond au *Nombre* contenu dans *zdtNombre*. Avec la même précaution quant au symbole décimal.

La source de *zdtValeurInf*:

```
=RechDom("Valeur";"Observations";"Nombre=" &
replace([zdtBorneSup];",",";"))
```

7.4. Recherche de la borne et de la valeur supérieures

Procédons par analogie.

La source de *zdtNombreSup*:

```
=MinDom("Nombre";"Observations";"Nombre>=" &
Replace([zdtNombre];",",";"))
```

La source de *zdtValeurSup*:

```
=RechDom("Valeur";"Observations";"Nombre=" &
replace([zdtBorneSup];",",";"))
```

7.5. Variations entre les deux bornes

Ici, il s'agit simplement d'opérer la différence du contenu des contrôles respectifs.

La source de *zdtDeltaBorne*:

```
= [zdtBorneSup] - [zdtBorneInf]
```

La source de *zdtDeltaValeur*:

```
= [zdtValeurSup] - [zdtValeurInf]
```

7.6. L'extrapolation proprement dite

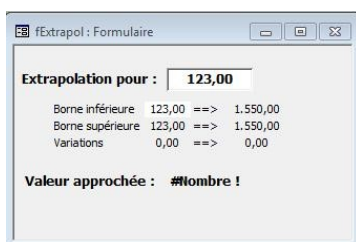
7.6.1. Cas banal : *zdtNombre* se situe entre deux bornes

Ici aussi, il s'agit d'une opération arithmétique : à la valeur de la borne inférieure, il faut ajouter la proportion de variation :

```
= [zdtValeurInf] + ([zdtDeltaValeur] * ([zdtNombre] -
[zdtBorneInf])) / [zdtDeltaBorne]
```

7.6.2. Cas particulier : *zdtNombre* est une borne

Dans ce cas :



En effet, *[zdtDeltaBorne]* est égal à zéro ce qui rend la division illégale !

Il faut donc exprimer dans la source de *zdtExtrapol* : si *zdtDeltaBorne = 0*, alors *ZdtExtrapol = zdtValeurInf*; sinon calculer l'extrapolation.

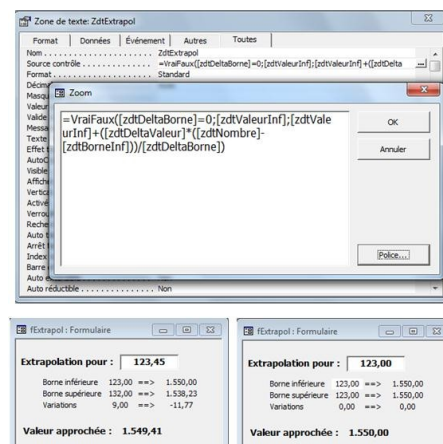
Schématiquement :

```
VraiFaux(<zdtDeltaBorne = zéro>;< ZdtExtrapol =
zdtValeurInf>;<extrapoler>)
```

C'est-à-dire :

```
=VraiFaux([zdtDeltaBorne]=0; [zdtValeurInf];
[zdtValeurInf] + ([zdtDeltaValeur] * ([zdtNombre] -
[zdtBorneInf])) / [zdtDeltaBorne])
```

Et voilà :



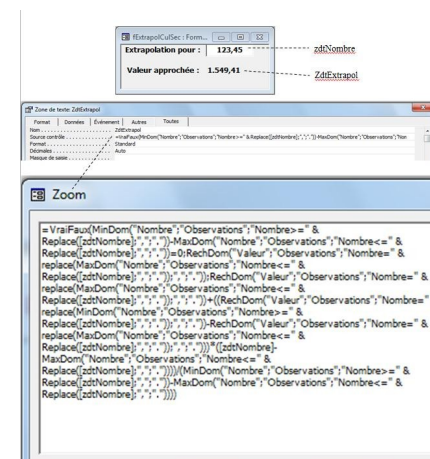
7.6.3. Gorgée après gorgée ou cul sec ?

Nous avons progressé pas à pas en détaillant chaque étape du raisonnement pour calculer l'extrapolation.

Ainsi, pour concrétiser chaque étape, nous avons créé des contrôles pour y loger les résultats intermédiaires.

S'il s'avère que ces renseignements sont inutiles pour l'utilisateur final, il suffit de cacher les contrôles en attribuant la valeur « Non » à leur propriété *Visible*.

Mais on peut aussi court-circuiter toutes les étapes intermédiaires.



Il va sans dire que cette source n'a pas été écrite à la volée.

En réalité, je suis parti de ceci :

```
=VraiFaux([zdtDeltaBorne]=0;[zdtValeurInf];  
[zdtValeurInf]+([zdtDeltaValeur]*([zdtNombre]-  
[zdtBorneInf]))/[zdtDeltaBorne])
```

et j'ai patiemment remplacé tous les [zdtxxxx] autres que [zdtNombre] par leur source (sauf le signe « = » initial).

Ce n'est pas nécessairement une bonne idée car c'est illisible, je l'ai fait pour montrer « simplement » que c'est possible.

8. Conclusion

Vous avez maintenant le pied à l'étrier pour l'emploi des fonctions intégrées.

Pour faire un pas de plus dans la découverte de cette matière, voyez le tutoriel de Philippe JOCHMANS : Les Fonctions de Domaine dans Access ([Lien 50](#)).

9. Base de données

Au format Access2000 : Lien [51](#).

Retrouvez l'article de Claude Leloup en ligne : [Lien 52](#).



Oracle publie la première liste du groupe d'experts chargé de Java 8, le support de la modularité reporté à Java 9

Mark Reinhold, architecte en chef du groupe de la plateforme Java chez Oracle, vient d'annoncer dans un billet de blog les premiers membres du groupe d'experts chargé de la future version du langage.

Ces experts sont des membres du JCP (Java Community Process) qui sont chargés de créer une première ébauche de la spécification JSR (Java Specification Request) 337 qui sera soumise à l'examen de la communauté.

Aux côtés de Mark Reinhold, travailleront Andrew Haley de Red Hat et Steve Poole d'IBM. Google a manifesté également son intention de participer au projet, mais la société n'a pas encore désigné son représentant qui intégrera le groupe.

Dans un autre article séparé, Mark Reinhold a annoncé le report de l'ajout du système de module à Java SE 9.

Ce report est dû au fait qu'Oracle souhaiterait respecter sa feuille de route pour Java qui prévoit la sortie des nouvelles versions du langage tous les deux ans. Or, l'ajout de la modularité va apporter des changements importants au langage, qui entraîneront des tests supplémentaires pouvant retarder la sortie de Java SE 8 à mi-2014.

Le support de la modularité initialement prévu à partir du projet Jigsaw ne sera donc pas au rendez-vous dans Java SE 8, et sera reporté à 2015, date à laquelle est prévue la sortie de Java 9.

La publication de la version finale de Java 8 est prévue en août 2013 et le JDK8 en septembre 2013.

Commentez la news de Hinault Romaric en ligne : [Lien 53.](#)

Les derniers tutoriels et articles

Simplifier l'écriture des listeners Java avec EventHandler

Ce tutoriel a pour objectif de vous présenter la classe `java.beans.EventHandler` qui permet de simplifier la création de listeners en Java.

1. Introduction

L'écriture d'IHM java mène généralement à l'écriture de plusieurs listeners, souvent sous la forme de classes anonymes. Celles-ci sont fastidieuses à écrire et réduisent la lisibilité du code pour au final ne faire qu'une simple affectation ou un simple appel de méthode.

La classe `java.beans.EventHandler` ([Lien 54](#)) permet de s'affranchir de cette écriture en permettant de générer dynamiquement des instances d'interface dont les méthodes exécutent une instruction simple. Son mécanisme est basé sur la création de proxy (cf. `java.lang.reflect.Proxy` ([Lien 55](#)), `java.lang.reflect.InvocationHandler` ([Lien 56](#))).

Son utilisation est plus efficace que les classes anonymes, notamment pour les grandes applications dans lesquelles une même interface est implémentée de nombreuses fois. Elle permet de réduire la taille et l'empreinte mémoire de l'application.

La raison de cette faible empreinte est que la classe proxy utilisée par l'EventHandler partage les implémentations pour des interfaces identiques. Par exemple, pour créer tous les `ActionListeners` d'une application, une seule implémentation sera créée par le proxy. Globalement, une implémentation est créée par interface alors qu'en utilisant des classes anonymes, une implémentation est créée pour

chaque listener.

Par exemple pour une application comptant 11 `ActionListeners` et 3 `MouseListeners`, 2 implémentations seront créées avec l'EventHandler (une pour les `ActionListeners` et une pour les `MouseListeners`) alors que l'utilisation de classes anonymes en aurait créé 14 (une pour chaque listener).

Son inconvénient est que l'utilisation de la réflexion rend l'exécution plus coûteuse. Ce coût est négligeable pour des exécutions ponctuelles (action d'un clic bouton par exemple) mais cela peut nuire aux performances dans le cas d'appels successifs (appels dans une boucle).

La classe `EventHandler` permet globalement de gérer plusieurs cas d'utilisation simples via ses méthodes statiques `create(...)`.

2. Appel simple d'une méthode

Le premier cas d'utilisation est celui d'un simple appel à une méthode donnée (l'**action**) sur un objet donné (la **cible**). Il est géré par la méthode statique `EventHandler.create(Class, Object, String)` ([Lien 57](#)).

Prenons un exemple simple :

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        foo();
    }
});
```

Ce code crée une instance de l'interface ActionListener afin d'effectuer un appel à la méthode foo() de l'objet englobant la classe anonyme.

Dans notre cas d'utilisation, la méthode foo() est l'action et l'objet englobant est la cible.

Avec la classe EventHandler, le code devient :

```
button.addActionListener(EventHandler.create(ActionListener.class, this, "foo"));
```

Le premier paramètre est le type de l'instance qui doit être créée.

Le deuxième paramètre est la cible sur laquelle sera effectuée l'action (ici this).

Le troisième paramètre est l'action à effectuer sur la cible (la méthode à exécuter, ici foo()).

Un autre exemple :

```
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        helloworld.sayHello();
    }
});
```

Devient :

```
button.addActionListener(EventHandler.create(ActionListener.class, helloworld, "sayHello"));
```

3. Appel d'une méthode avec une propriété de l'évènement comme argument

L'EventHandler permet également de pouvoir spécifier à l'action une des propriétés de l'évènement avec la méthode statique EventHandler.create(Class, Object, String, String) ([Lien 58](#)).

Soit le code :

```
new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        doAction(e.getSource());
    }
};
...
public void doAction(Object object) {
    System.out.println("action sur " + object);
}
```

Qui, avec l'EventHandler devient :

```
EventHandler.create(ActionListener.class, this, "doAction", "source");
...
public void doAction(Object object) {
    System.out.println("action sur " + object);
}
```

L'utilisation des propriétés nécessite la présence des getters appropriés, la propriété "source" implique la présence de la méthode getSource().

Il est également possible d'indiquer une sous-propriété en utilisant le caractère ".". Ainsi le code suivant utilisant la propriété text de la propriété source de l'évènement :

```
new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        setTitle(((JTextField)
e.getSource()).getText());
    }
};
```

Devient avec l'EventHandler :

```
EventHandler.create(ActionListener.class, this, "setTitle", "source.text");
```

Il est possible d'enchaîner les sous-propriétés. Par exemple :

```
EventHandler.create(ActionListener.class, this, "setAdminMode", "source.user.admin");
```

est équivalent à :

```
new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        setAdminMode(((UserAuth)
e.getSource()).getUser().isAdmin());
    }
};
```

Il est aussi possible de passer l'évènement lui-même en spécifiant une chaîne vide "" comme propriété :

```
EventHandler.create(ActionListener.class, this, "doAction", "");
...
public void doAction(ActionEvent e) {
    System.out.println("action sur " +
e.getSource());
}
```

4. Choix des méthodes de l'interface

Par défaut, le code expliqué précédemment s'applique à toutes les méthodes de l'interface. Par exemple le code :

```
EventHandler.create(FocusListener.class, this, "focusChanged", "");
```

est équivalent à :

```

new FocusListener() {

    @Override
    public void focusGained(FocusEvent e) {
        focusChanged(e);
    }

    @Override
    public void focusLost(FocusEvent e) {
        focusChanged(e);
    }
};

```

Pour n'appliquer une action qu'à une méthode précise de l'interface, il est possible de spécifier son nom avec la troisième méthode statique `EventHandler.create(Class, Object, String, String, String)` ([Lien 59](#)). Les autres méthodes ne feront alors rien.

```

EventHandler.create(FocusListener.class, this,
"focusChanged", "", "focusGained");

```

est équivalent à :

```

new FocusListener() {

    @Override
    public void focusGained(FocusEvent e) {
        focusChanged(e);
    }

    @Override
    public void focusLost(FocusEvent e) {
    }
};

```

5. Cas de la méthode cible surchargée

Il est recommandé de ne pas appeler une méthode surchargée comme action. En effet, si par exemple la cible est une instance de la classe suivante :

```

public class Target {
    public void doAction(String);
    public void doAction(Object);
}

```

L'EventHandler appellera la méthode appropriée en fonction de la source. Cependant, si la source vaut null, les deux méthodes sont appropriées et il n'y a aucune garantie de laquelle sera appelée. Pour cette raison il est déconseillé d'appeler une méthode surchargée comme action.

6. Conclusion

Même si l'EventHandler ne couvre que des cas d'utilisation relativement basiques, ceux-ci représentent une grande majorité des cas rencontrés dans une application. De plus, même si son utilisation est particulièrement adaptée aux listeners, elle peut s'utiliser pour créer une instance de n'importe quelle interface.

```

EventHandler.create(Runnable.class, "execute");

```

Ainsi son principal intérêt est donc de grandement réduire le nombre de classes anonymes ce qui apporte une plus grande lisibilité du code en plus d'une plus faible empreinte mémoire.

Retrouvez l'article de Yann D'Isanto en ligne : [Lien 60](#)



La fondation Eclipse sort Eclipse Juno 3.8 /4.2, une double version de l'EDI riche en nouveautés

La fondation Eclipse vient de sortir la version Juno d'Eclipse.

Une nouvelle version d'Eclipse est disponible. Elle porte le nom de Juno. Cette version est en fait une double version, puisque nous avons droit à la fois à la version 3.8 et à la version 4.2.

Il faut savoir que la version 3.8 est la dernière "version" pour Eclipse 3.XX. Des versions de maintenance sont prévues pour la 3.8, mais aucune version majeure supérieure n'est prévue dans les 3.XX.

Cela tient d'une volonté de la fondation Eclipse de basculer majoritairement sur les versions 4.XX. La version 4 (précédemment baptisée e4) d'Eclipse est en développement depuis quatre ans et constitue la majeure partie de cette release.

Des informations supplémentaires sur les nouveautés de cette version sont disponibles à cette adresse : notes pour la version 3.8 : [Lien 61](#), notes pour la version 4.2 : [Lien 62](#).

Pour télécharger cette nouvelle version rendez-vous sur la page de téléchargement d'Eclipse : [Lien 63](#)

D'un point de vue esthétique, cette version inclut :

- une barre de recherche rapide ;
- un nouveau look pour le workbench ;
- la possibilité de mélanger les vues et les éditeurs (plus de zone réservée aux seuls éditeurs) ;
- la possibilité de détacher les éditeurs.

De plus profonds changements ont été apportés au modèle de programmation. Le workbench est maintenant représenté comme un modèle EMF et est rendu dynamiquement.

Les composants graphiques peuvent maintenant être customisés à l'aide d'un moteur CSS.

La plateforme supporte maintenant l'injection de dépendances.

L'incubateur e4 comporte quelques outils absolument intéressants (comme un éditeur de modèle qui permet le changement "en live" de l'interface graphique ainsi qu'un CSS-Spy pour examiner les propriétés CSS des éléments).

Enfin, cette nouvelle version inclut une couche de compatibilité qui permet de supporter les versions 3.XX sur cette plateforme.

Dans les nouveautés, nous pouvons également citer :

- le plug-in de recommandation de code : ce plug-in va indiquer au développeur, dans l'assistant de contenu, les méthodes les plus probables à implémenter, des templates de codes le plus souvent utilisés (ex. pour la création d'un *Button*, on lui donne un texte, un layout, des styles puis une action), des recommandations sur les méthodes les plus surchargées, une documentation étendue, une complétion plus complexe pour les mots contenus dans la méthode (ex. : "background" pour "drawBackground()" ou "getBackgroundImage()" ;
- XText/Xtend (support du débogage intégré des DSL) ;
- Orion ;
- des améliorations pour JDT notamment : au niveau du support de Java 7, des améliorations du moteur d'évaluation (pour les breakpoints conditionnels), de l'analyse basée sur les annotations... ;
- amélioration du support de GiT ;
- Virgo : un projet qui embarque un noyau Nano pour créer de petites applications web basées sur OSGi ;
- Koneki : projet qui fournit un IDE pour Lua (un langage de script) ;
- amélioration RAP ;
- utilisation des features pour lancer des configurations et des cibles ;
- Mylyn Intent.

La liste des nouveautés se trouve ici : [Lien 64](#)

Commentez la news de Gueritarish en ligne : [Lien 65](#).



NetBeans 7.2 disponible : améliorations de performances et de la productivité avec le support de PHP 5.4, C++ 11, FindBugs et Smarter

NetBeans, l'environnement de développement intégré (EDI) open source pour Java, PHP, C et C++ développé par Oracle vient de passer officiellement à la version 7.2 stable, six mois seulement après la sortie de la version 7.1.

NetBeans 7.2 a été conçu pour offrir aux développeurs une expérience plus fluide et de meilleures performances lors de l'écriture du code.



Les développeurs de l'EDI se sont concentrés autour de trois axes majeurs :

- amélioration de la productivité du développeur : NetBeans 7.2 intègre par défaut l'extension FindBugs, un outil pour l'analyse statique du code Java. L'ajout de FindBugs permettra aux développeurs de trouver rapidement des bugs dans leur code ;
- amélioration de performances : l'introduction du projet Smarter permet la numérisation des projets en arrière-plan, rendant ainsi le démarrage de l'EDI plus rapide ;
- support élargi : Netbeans 7.2 prend en charge PHP 5.4, Symfony2, Doctrine2, Maven 3.0.4, Groovy 1.8.6, le standard C++ 11 et JavaFX 2.1.1.

NetBeans 7.2 est téléchargeable gratuitement pour les systèmes d'exploitation Windows, Mac OS X, Linux et Solaris.

Commentez la news de Hinault Romaric en ligne : [Lien 66](#)



Google publie le SDK d'Android 4.1 Jelly Bean, avec de nouvelles API

Google vient de publier le SDK (kit de développement) pour Android 4.1 Jelly Bean.

Jelly Bean est la dernière version du système d'exploitation mobile de Google qui vient avec son lot de nouveautés (lire ci-avant). La société avait publié le code source de l'OS ainsi que les binaires pour certains dispositifs il y a de cela une dizaine de jours.

Les développeurs peuvent désormais profiter du plein potentiel de cette mise à jour de l'OS mobile dans leurs applications à travers le nouveau SDK Android, qui apporte de nouvelles API ainsi qu'un nouvel émulateur pour les tests.

La tablette Nexus 7 est l'un des premiers dispositifs à utiliser Jelly Bean et, dans un billet de blog, Google recommande aux développeurs de mettre leurs applications à jour pour l'écran sept pouces haute définition de l'appareil, afin que celles-ci s'exécutent parfaitement.

Le SDK Android 4.1 peut être téléchargé à partir de votre SDK Manager.

Télécharger le SDK Android 4.1 : [Lien 67](#)

Commentez cette news de Hinault Romaric en ligne : [Lien 68](#)

Flash ne sera pas disponible sur Android 4.1, Adobe annonce le retrait de l'application de Google Play dès le 15 août

Le lecteur Flash ne sera pas disponible pour Android 4.1, la prochaine version du système d'exploitation mobile présentée la semaine dernière lors du Google I/O.

Adobe avait annoncé en fin d'année dernière qu'il arrêterait avec le développement de Flash pour les plateformes mobiles (lire ci-avant).

C'est donc sans surprise que l'éditeur dans un billet de blog a déclaré qu'il n'y aura pas de certification de Flash pour Android 4.1. Le programme de certification permet à Adobe de travailler avec les constructeurs pour s'assurer que le lecteur fonctionne correctement lorsqu'il est préinstallé par défaut sur un dispositif.

Par ailleurs, à compter du 15 août, il ne sera plus possible de télécharger et installer l'application depuis la galerie Android (Google Play).

Les mises à jour seront néanmoins disponibles uniquement pour les terminaux ayant déjà une version de Flash installée. La société conseille donc aux utilisateurs d'Android 4.0 ou versions antérieures souhaitant utiliser son lecteur de l'installer avant le 15 août.

Adobe recommande également pour toute mise à jour vers Android 4.1, de désinstaller l'application Flash qui n'a pas été testée avec cette version du système d'exploitation.

Commentez cette news de Hinault Romaric en ligne : [Lien 69](#)

Android : les API ne peuvent être soumises au Copyright, décide le juge chargé de l'affaire, Oracle compte faire appel

Le procès entre Oracle et Google est terminé. En tout cas le premier procès.

Après avoir été débouté de quasiment toutes ses demandes, Oracle avait demandé au juge un nouveau procès pour revoir les faits et les premières décisions. Dans un document publié en fin de semaine dernière le juge refuse cette possibilité au motif qu'Oracle n'apporte aucun argument nouveau dans sa demande.

Oracle accusait Google d'avoir enfreint plusieurs de ses brevets en utilisant certaines API dans Android. La société de Larry Ellison affirmait également que les ingénieurs de Google avaient en toute connaissance de cause enfreint ces brevets et qu'ils avaient même recopié du code copyrighté dans l'écriture de l'OS.

Le juge en a décidé autrement d'une part en statuant que les API ne pouvaient être ni brevetées ni copyrightées et d'autre part en constatant que le code « copié » ne concernait que 9 lignes d'une opération particulièrement triviale.

Cette décision avait d'ailleurs été l'occasion d'un coup de théâtre ([Lien 70](#)) puisque le juge avait alors fait savoir qu'il était lui-même développeur à ses heures perdues et que – en substance – Oracle avait intérêt à ne pas trop le prendre pour un incompetent en informatique.

Si ce refus de révision marque la fin de la partie, elle ne marque pas pour autant la fin de la bataille juridique entre les deux sociétés. Oracle a encore la possibilité de faire appel auprès d'une Cour dont les compétences sont jugées supérieures au « District Cour » par le droit américain pour faire casser ce jugement.

C'est sans surprise l'option que choisira Oracle. Qui doit secrètement espérer que le prochain juge chargé de

l'affaire sera plus coulant et peut-être un peu moins développeur que le juge Alsup.



William Alsup, juge dans l'affaire Oracle vs Google

Google demande à présent qu'Oracle lui rembourse les frais de justice engagés pour sa défense. Une demande sur laquelle le juge Alsup devra se prononcer avant de laisser cette affaire à un confrère.

Commentez cette news de Gordon Fowler en ligne : [Lien 71](#)

Liens

- Lien 01 : <http://inserthtml.developpez.com/tutoriels/javascript/interactivite-avec-balise-html5-canvas/fichiers/>
- Lien 02 : <http://inserthtml.developpez.com/tutoriels/javascript/interactivite-avec-balise-html5-canvas/fichiers/demo.zip>
- Lien 03 : <http://www.inserthtml.com/2012/05/html5-canvas-interactivity/>
- Lien 04 : <http://inserthtml.developpez.com/tutoriels/javascript/interactivite-avec-balise-html5-canvas/>
- Lien 05 : <http://csharpindepth.com/Articles/General/Strings.aspx>
- Lien 06 : <http://www.unicode.org/>
- Lien 07 : <http://tlevesque.developpez.com/traductions/passage-parametres-csharp/>
- Lien 08 : <http://www.yoda.arachsys.com/csharp/memory.html>
- Lien 09 : <http://csharpindepth.com/Articles/General/Unicode.aspx>
- Lien 10 : <http://msdn.microsoft.com/en-us/library/ms973919>
- Lien 11 : <http://jormes.developpez.com/traductions/strings-csharp-dotnet/>
- Lien 12 : <http://ra3s.com/wordpress/dysfunctional-programming/return-code-vs-exception-handling/>
- Lien 13 : <http://blogs.msdn.com/oldnewthing/archive/2005/01/14/352949.aspx>
- Lien 14 : <http://cpp.developpez.com/livres/?page=tous#L0201615622>
- Lien 15 : <http://cpp.developpez.com/livres/?page=tous#L0321334876>
- Lien 16 : <http://alexandre-laurent.developpez.com/cpp/retour-fonctions-ou-exceptions/>
- Lien 17 : <http://www.developpez.com/actu/44956/Microsoft-va-t-il-devoir-racheter-Nokia-pour-perenniser-Windows-Phone/>
- Lien 18 : <http://www.agile-workers.com/web/2012/06/leadership-strategy-and-qt/>
- Lien 19 : <http://press.nokia.com/2012/06/14/nokia-sharpens-strategy-and-provides-updates-to-its-targets-and-outlook/>
- Lien 20 : <http://www.developpez.net/forums/d1042408/c-cpp/bibliotheques/qt/nokia-vend-division-qt-digia/#post5823941>
- Lien 21 : <http://qt.developpez.com/actu/46153/De-l-avenir-des-Ot-Developer-Days-toujours-pas-d-annonce-officielle-la-communaute-reprend-les-choses-en-main/>
- Lien 22 : <http://www.developpez.net/forums/d672056/c-cpp/bibliotheques/qt/qt-4-5-sous-licence-lgpl/>
- Lien 23 : <http://www.developpez.net/forums/d811427/c-cpp/bibliotheques/qt/qt-quick/qt-quick-futur-developpement-dihm-anciennement-declarative-ui/>
- Lien 24 : <http://qt.developpez.com/actu/42931/Sortie-de-Ot-4-8-1-Au-menu-corrections-de-bogues-et-support-commercial-de-plusieurs-RTOS/>
- Lien 25 : <http://www.developpez.net/forums/d1251510/c-cpp/bibliotheques/qt/digia-sinvestit-plus-plus-qt-nokia-continue-t-somber/>
- Lien 26 : <http://www.developpez.net/forums/f1563/c-cpp/bibliotheques/qt/outils/bibliotheques/qxorm/>
- Lien 27 : <http://qt.developpez.com/faq/>
- Lien 28 : <http://qt.developpez.com/tutoriels/mvc/apostille-delegates-mvd/>
- Lien 29 : <http://louis-du-verdier.developpez.com/qt/fondations/>
- Lien 30 : <https://gitorious.org/patience>
- Lien 31 : <http://dl.dropbox.com/u/28758727/Patience.zip>
- Lien 32 : <http://tcuvelier.developpez.com/tutoriels/qt/mobile-meego/qml/premiere-application/>
- Lien 33 : <http://louis-du-verdier.developpez.com/qt/qml/communication/>
- Lien 34 : https://gitorious.org/patience/patience/trees/Chap_II-A-3
- Lien 35 : <http://louis-du-verdier.developpez.com/qt/fondations/>
- Lien 36 : https://gitorious.org/patience/patience/trees/Chap_II-B-3
- Lien 37 : https://gitorious.org/patience/patience/trees/Chap_II-B-4
- Lien 38 : https://gitorious.org/patience/patience/trees/Chap_III-B
- Lien 39 : https://gitorious.org/patience/patience/trees/Chap_III-C-3
- Lien 40 : https://gitorious.org/patience/patience/trees/Chap_III-E
- Lien 41 : <http://sfabry.developpez.com/tutoriels/qt/designer-avec-qml-jeu-patience/>
- Lien 42 : <http://get.qt.nokia.com/developerguides/qtquickdesktop/OtQuickApplicationGuide4Desktop.pdf>
- Lien 43 : http://qt.ftp.developpez.com/tutoriels/qt/guide-developpement/application-bureau-avec-qt-quick/fichiers/notezapp_src.zip
- Lien 44 : <http://doc-snapshot.qt-project.org/qtcreator-2.5/quick-projects.html#creating-qt-quick-ui-projects>
- Lien 45 : <http://doc-snapshot.qt-project.org/qtcreator-2.5/quick-components.html>
- Lien 46 : <http://qt-project.org/doc/qt-4.8/qml-anchor-layout.html>
- Lien 47 : <http://qt-project.org/doc/qt-4.8/qml-positioners.html>
- Lien 48 : <http://qt-project.org/doc/qt-4.8/qml-item.html#z-prop>
- Lien 49 : <http://qt.developpez.com/tutoriels/qt/guide-developpement/application-bureau-avec-qt-quick/>
- Lien 50 : <http://starec.developpez.com/tuto/fonctionsdomaines/>
- Lien 51 : <http://claudeleloup.developpez.com/tutoriels/access/consulter-table-paliers/TablePaliers.mdb>
- Lien 52 : <http://claudeleloup.developpez.com/tutoriels/access/comment-trouver-valeur-dans-table-qui-contient-paliers/>
- Lien 53 : <http://www.developpez.net/forums/d1244798/java/general-java/oracle-publie-premiere-liste-groupe-d-experts-charge-java-8-a/>

- Lien 54 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/beans/EventHandler.html>
- Lien 55 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/lang/reflect/Proxy.html>
- Lien 56 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/lang/reflect/InvocationHandler.html>
- Lien 57 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/beans/EventHandler.html#create%28java.lang.Class,%20java.lang.Object,%20java.lang.String%29>
- Lien 58 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/beans/EventHandler.html#create%28java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String%29>
- Lien 59 : <http://javasearch.developpez.com/j2se/1.6.0/docs/api/java/beans/EventHandler.html#create%28java.lang.Class,%20java.lang.Object,%20java.lang.String,%20java.lang.String,%20java.lang.String%29>
- Lien 60 : <http://ydisanto.developpez.com/tutoriels/java/eventhandler/>
- Lien 61 : http://www.eclipse.org/eclipse/development/readme_eclipse_3.8.html
- Lien 62 : http://www.eclipse.org/eclipse/development/readme_eclipse_4.2.html
- Lien 63 : <http://www.eclipse.org/downloads/>
- Lien 64 : http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2FwhatsNew%2Fplatform_whatsnew.html
- Lien 65 : <http://www.developpez.net/forums/d1237993/environnements-developpement/eclipse/eclipse-juno-disponible/>
- Lien 66 : <http://www.developpez.net/forums/d1226662/java/edi-outils-java/netbeans/oracle-publie-netbeans-7-2-beta/>
- Lien 67 : <http://developer.android.com/sdk/index.html>
- Lien 68 : <http://www.developpez.net/forums/d1238077/java/general-java/java-mobiles/android/android-4-1-debarque-google-i-o-preview-sdk-disponible/>
- Lien 69 : <http://www.developpez.net/forums/d1150770/webmasters-developpement-web/flash-flex/fin-flash-player-mobile-adobe-veut-concentrer-efforts-technologie-air/>
- Lien 70 : <http://www.developpez.com/actu/44331>
- Lien 71 : <http://www.developpez.net/forums/d1211211/java/general-java/java-mobiles/android/proces-entre-google-oracle-autour-dandroid-commence/>