1. Reflector est un outil capable de décompiler n’importe quelle assembly .NET, que ce soit une DLL ou un exécutable. C’est l’un des outils les plus indispensables dans la panoplie de tout bon développeur .NET. BAMLViewer est un complément à Reflector apte à extraire le contenu binaire des fichiers BAML et d’en afficher le contenu XAML.
2. Le langage XAML applique les règles suivantes pour tout ce qui concerne la gestion des espaces et des blancs :
   * les retours à la ligne entre les caractères « East Asian » sont supprimés ;
   * les espaces, les sauts de lignes et les tabulations sont convertis en espace ;
   * les espaces consécutifs sont supprimés et remplacés par un seul espace ;
   * un espace situé immédiatement après la balise d’ouverture d’un élément est supprimé ;
   * un espace situé immédiatement avant la balise de fin d’un élément est supprimé.
   * Il est parfois nécessaire d’insérer plusieurs espaces ou de conserver un retour à la ligne

sur un contrôle. Pour parvenir à vos fins, vous pouvez utiliser le code suivant où, pour

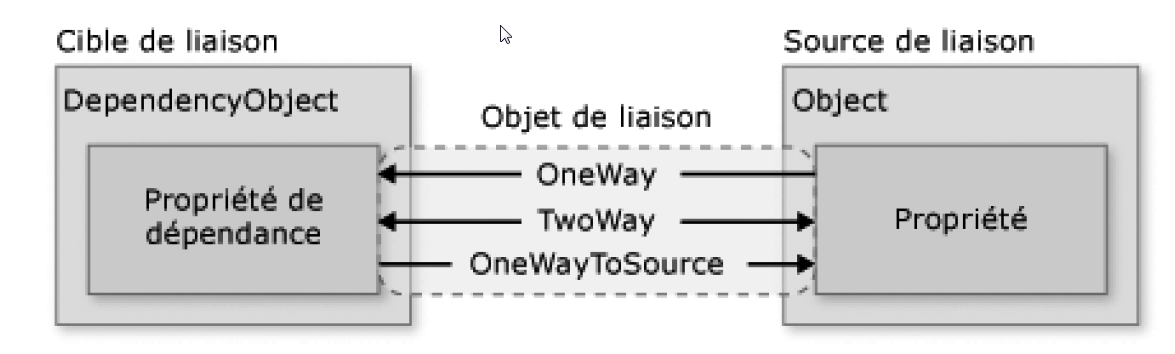
chacun des éléments dont on veut préserver les espaces et sauts de ligne, on définit un

attribut :xml:space="preserve". Cet attribut permet à vos contrôles de conserver les espaces (et les blancs) que vous avez définis.

1. Les controleurs : voir <http://www.areaprog.com/wpf/cours-448-les-layout-controls-description-des-differents-layout-controls> (Les layout controls - WPF.pdf)
   * Canvas : permet de positionner un élément avec les cordonnées x,y
   * stackPanel permet d’empiler les objets, les uns sur les autres. Il présente l’inconvénient de tronquer les éléments attachés si ils dépassent la taille. Microsoft propose un autre contrôleurs pour pallier à ce problème : *WrapPanel*
   * Grid. Il est conceptuellement très similaire à un tableau HTML : il permet d’afficher les informations dans une structure composée de lignes et colonnes. Pour pouvoir employer ce contrôle, il convient tout d’abord de déclarer les différentes lignes et colonnes qui composeront la grille. Pour cela, il faut utiliser les collections RowDefinitions et ColumDefinitions.
     1. Lorsqu’il devient nécessaire d’insérer des contrôles dans cette grille, il suffit d’indiquer dans quelle ligne et quelle colonne chacun des contrôles se situe : <Button Content="Bouton dans grille" Grid.Row="1" Grid.Column="1" />.
     2. Il est également possible de fusionner l’insertion des contrôles sur plusieurs lignes et plusieurs colonnes. Il faut alors passer par les propriétés RowSpan et ColumnSpan, qui prennent en paramètre le nombre de lignes et de colonnes sur lesquelles le contrôle doit être inséré :<Button Content="Bouton dans grille" Grid.Row="1" Grid.ColumnSpan="2" />
   * WPF propose six controleurs qui permettent de faire des dessins : line, ellipse …(voir Contrôles de dessin dans wpf par la pratique)
2. Les content controls : Les content controls sont des contrôle qui héritent tous de la class ContentControl et ils ne peuvent contenir que un seul élément (voir : <http://www.areaprog.com/wpf/cours-476-les-content-controls-description-des-differents-content-controls>)
   * Menu : Les \_ dans les valeurs des headers signifient que la lettre juste après est la touche pour accéder à cet MenuItem en appuyant sur Alt au préalable. On peut rajouter un separateur entre les menuitems : <Separator/>
3. Les item controls : Un item control est un contrôle qui peut avoir plus d'un élément enfant (Child item). Typiquement une ListeBox , ListView , GridView , …(voir http://www.areaprog.com/wpf/cours-479-les-item-controls-description-des-differents-item-controls)
4. Les ressources et les styles : <http://www.areaprog.com/wpf/cours-512-ressources-et-styles-personnalisation-d-interface-graphique>
   * dictionnaire de ressources : sert à externaliser les ressources dans un fichier à par(.xaml)
5. Les modèles de contrôle : les styles sont loin de nous apporter autant que les Template. En effet les styles comme on l’a vu vont nous permettent de modifier l’apparence
6. d’un ensemble de contrôles de par leurs propriétés. Alors que les Template vont totalement nous permettre de changer le modèle d’un contrôle
7. **La liaison de données** (data binding) : est une technique qui met en relation des données métier et une interface utilisateur. Chaque liaison est composée de quatre éléments :
   * un objet cible ;
   * une propriété cible ;
   * une source de liaison ;
   * une propriété source.

Par exemple, si l’on désire lier le contenu d’une TextBox à la propriété Nom d’un objet de type Client, l’objet cible sera la TextBox, la propriété cible sera la propriété Text, la valeur à utiliser sera Nom et l’objet source sera de type Client

1. Options de la liaison de données : le flux de données d’une liaison peut être dirigé de la cible vers la source (la valeur de la source change lorsque l’utilisateur modifie une valeur dans la TextBox) et/ou de la source vers la cible. Il est possible, via la propriété Mode d’un objet de type Binding, d’indiquer de quelle façon un objet métier ou un contrôle graphique doit être mis à jour, comme le montre la figure ci-dessous :



1. Data binding par xaml :
   * spécifier une source de liaison
   * Utiliser la propriété DataContext d’un élément parent est utile uniquement lorsque plusieurs propriétés sont liées à la même source.
   * En liant un élément graphique à un autre élément graphique, il est possible de recourir aux propriétés ElementName ou RelativeSource pour indiquer la source de la liaison.
   * Exemple (voir projet BindingParXaml) : lier la largeur d’un bouton à la valeur d’un curseur (slider)
   * Si la source de la liaison est un objet, il faut utiliser la propriété Path pour en préciser la valeur. Cependant, si cette liaison s’établit entre l’interface graphique et une source de données XML, alors il faudra indiquer la propriété XPath.
2. Data binding code csharp :
   * Déclarer un objet de type Binding
   * Exemple (voir projet BindingParCode) : afficher une liste de contact. Nous disposons d’une ListBox qui doit être liée au contenu d’un fichier XML.
   * Étant donné que nous avons lié la propriété DataContext à la liste de contacts, il suffit de « récupérer » cette liste et de l’affecter à la propriété ItemsSource : ItemsSource="{Binding}"
   * Ce code utilise une syntaxe de liaison de données vide, c’est-à-dire sans préciser de propriété Path, Source ou ElementName : {Binding}. La propriété DataContext de la ListBox est liée à la liste de contacts. Lorsque la propriété Path n’est pas spécifiée, le moteur WPF effectue la liaison sur l’objet entier.
3. Data binding : <http://slouge.wordpress.com/2009/09/05/wpf-first-step-into-binding/>
4. Validation de données :
   * Avec le framework .NET 3.5, le moteur de binding ajoute le support de la validation de la couche métier grâce à l’utilisation de l’interface IDataErrorInfo
   * Pour mettre en place voir wpf par la pratique page 105
   * Voir exemple WPFValidation
5. Créer son propre control, voir le Comment créer votre propre contrôle avec WPF .pdf ou chapitre Création de contrôles dans wpf par la pratique(exemple CreationControlesWPF)
6. Evènements
   * Bubbling event : Le bubble Event est un event qui ne sera pas uniquement traité par le contrôle qui l’a envoyé. Il va remonter le visual tree, c'est-à-dire la hiérarchie des contrôles graphiques(Exemple BubblingEventWpf)
   * Tunneling Events : Les tunneling Events sont le contraire des bubbling Events. Ces évènements sont d’abord raised par l’élément root (le plus haut) du visual tree vers l’élément source (Exemple TunnelingEventWpf). Pour arrêter le parcours d’un event, il faut mettre e.Handler = true ;
   * Créer un nouveau RoutedEvent : voir « Les RoutedEvents et les commandes en WPF.pdf » ( Exemple CreateRoutedEventWpf)
7. Commandes :
   * une commande est un objet static qui permet de rendre plus automatique et facile la programmation d’évènement répétitif.
   * Pour mettre en place nune commande, voir « Les RoutedEvents et les commandes en WPF.pdf » ( Exemple CommandWpf)
   * La cible est l’élément sur lequel la commande est exécutée. La source de la commande peut choisir la cible. Si la cible n’est pas définie, alors c’est le contrôle possédant le focus qui sera utilisé comme cible de la commande.