Mettre à jour une base de données : la gestion des commandes fournisseur

5

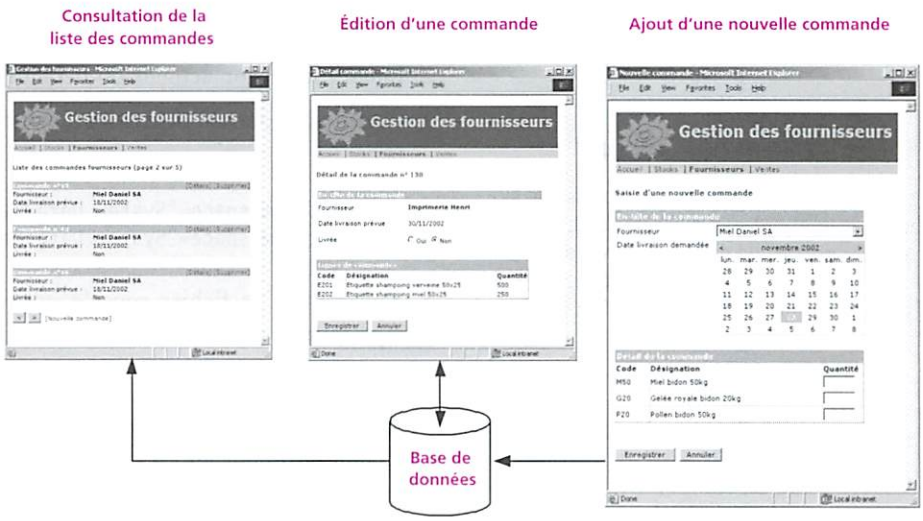
ASP.NET

Repeater | Event bubbling | PagedDataSource | SqlDataAdapter | SqlCommandBuilder | TemplateColumn

SOMMAIRE

MOTS-CLÉS

- Repeater
- Event bubbling
- PagedDataSource
- SqlDataAdapter
- SqlCommandBuilder
- TemplateColumn
- Contrôles de validation
- SqlTransaction



Dans ce chapitre, nous présentons les mécanismes de mise à jour d'une base de données à travers l'édition, l'ajout et la suppression de commandes fournisseur. À cette occasion, nous abordons également les techniques de validation des saisies effectuées par l'utilisateur, la gestion des transactions et la réalisation d'une liste personnalisée de commandes avec mise en œuvre de la pagination.

Affichage de la liste des commandes fournisseur

Le module de gestion des fournisseurs doit permettre la consultation, l'édition, l'ajout et la suppression de commandes fournisseur ; il sera constitué de trois pages distinctes :

- La page `Fournisseurs.aspx`, écran d'accueil du module, présentera la liste des commandes fournisseur.
- La page `DetailCommande.aspx` permettra de consulter le détail d'une commande (en-tête et lignes de commandes) et, le cas échéant, de modifier le statut de la livraison (livrée : oui/non).
- La page `NouvelleCommande.aspx` permettra de saisir une nouvelle commande, avec validation des informations fournies par l'utilisateur.

Avant de commencer, nous allons sauvegarder deux copies de la page `Fournisseurs.aspx` réalisée précédemment, qui nous servira de point de départ pour les développements de ce chapitre ; nous allons également ajouter les directives `<% Import...%>` nécessaires à la manipulation des données.

1. Démarrez Web Matrix.
2. Ouvrez la version actuelle du fichier `Fournisseurs.aspx`.
3. Placez-vous dans l'onglet All et rajoutez les directives suivantes en haut de la page :

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.SqlClient" %>
```

4. Sauvegardez le fichier sous le nom `DetailCommande.aspx`, dans le même répertoire.
5. Sauvegardez le fichier sous le nom `NouvelleCommande.aspx`, dans le même répertoire.

Ceci étant fait, nous allons passer à la réalisation de la page qui affiche la liste des commandes, basée sur l'utilisation d'un composant qui permet de réaliser des listes personnalisées : le contrôle `Repeater`.

RAPPEL Bibliothèque ADO.NET

Pour pouvoir faire référence aux classes de la bibliothèque ADO.NET par leur nom court, il faut importer les espaces de nommage `System.Data` pour les classes génériques et `System.Data.SqlClient` pour les classes d'accès à SQL Server.

Réalisation de la maquette de la liste des commandes avec Repeater

Comme pour toute page ASP.NET, nous allons commencer par la réalisation de la maquette : le cahier des charges est d'afficher, pour chaque commande, un résumé synthétique qui présente le nom du fournisseur, la date de livraison prévue, le statut de la livraison et comporte deux boutons Détails et Supprimer permettant respectivement d'éditer ou de supprimer la commande (voir figure 5-1).

La réalisation de cette page repose en grande partie sur l'utilisation du contrôle serveur `Repeater` qui, comme nous allons le voir, permet d'afficher le contenu d'une source de données suivant une maquette HTML définie par l'utilisateur.

Figure 5–1 La page de consultation de la liste des commandes

Voici la marche à suivre pour la réalisation de la maquette :

1. Démarrez Web Matrix et ouvrez le fichier `Fournisseurs.aspx` créé précédemment.
2. Placez-vous dans l'onglet Design de Web Matrix.
3. Insérez un titre « Liste des commandes fournisseur ».
4. Insérez un contrôle serveur `Repeater` depuis la barre d'outils Web Controls et renommez-le `ListeCommandes`.
5. Insérez un contrôle serveur de type `Label` affichant le texte « Il n'y a aucune commande dans cette liste », qui sera affiché en cas de liste vide, et renommez-le `ListeVide`.
6. Insérez un lien « NouvelleCommande » pointant vers la page `NouvelleCommande.aspx`.

Le résultat de la maquette doit ressembler à l'illustration de la figure 5-2.

Voici le contenu HTML correspondant (les contrôles serveur sont indiqués en couleur) :

Fournisseurs.aspx (partie graphique)

```
...
    <form runat="server">
        <h4>Liste des commandes fournisseur</h4>
        <asp:Repeater id="ListeCommandes" runat="server">
        </asp:Repeater>
```

Le contrôle serveur Repeater

Le contrôle `Repeater` permet d'afficher les éléments d'une source de données en utilisant un patron HTML (*template*) répété (et personnalisé) pour chaque occurrence. Web Matrix ne propose pas d'assistant graphique pour les contrôles de ce type : le développeur doit saisir directement le code dans l'onglet HTML.

Figure 5–2 Maquette de la page de consultation des commandes

```
        <p></p>
        <asp:Label id="ListeVide" runat="server">
        Il n'y a aucune commande dans cette liste
        </asp:Label>
        <p><a href="NouvelleCommande.aspx">[Nouvelle commande]</a></p>
        <p></p>
    </form>
...
```

Il nous reste maintenant à paramétrer le contrôle `Repeater`, pour qu'il affiche la liste des commandes : c'est ce que nous allons faire dans la section suivante.

Paramétrage du contrôle Repeater

Le contrôle serveur `Repeater` permet, comme son nom l'indique, de répéter plusieurs fois un même contenu HTML en liant chaque élément (`Item`) à un enregistrement de la source de données associée au contrôle (voir figure 5-3).

Il est possible de spécifier optionnellement un en-tête (`Header Item`), un pied (`FooterItem`) et un séparateur (`SeparatorItem`).

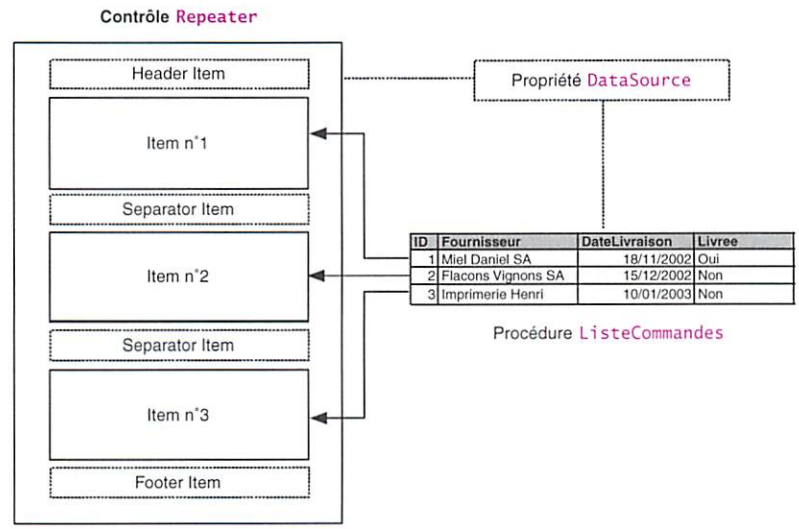


Figure 5–3 Architecture du contrôle Repeater

Le paramétrage du contrôle `Repeater` s'effectue donc en deux temps :

1. Association du contrôle à une source de données.
2. Réalisation de la maquette HTML des éléments.

Nous allons réaliser ces deux étapes dans l'ordre.

Liaison du contrôle Repeater à une source de données

On réalise la liaison entre un contrôle `Repeater` et une source de données par l'intermédiaire de la propriété `DataSource`, qui doit pointer vers une liste de données énumérable (`DataView`, `DataReader`, `ArrayList`...) : en l'occurrence, nous allons lier notre contrôle à la procédure `ListeCommandes`.

Le fonctionnement étant similaire à celui du contrôle `DataGrid` étudié au chapitre précédent, nous présentons directement le code correspondant, à implémenter dans l'onglet Code de Web Matrix :

Fournisseurs.aspx (Version C#)

```
void Page_Load(Object sender, EventArgs e)
{
    SqlConnection myConnection;
    myConnection = (SqlConnection)Session["myConnection"];

    string SQL = "ListeCommandes";
    SqlDataAdapter myAdapter = new SqlDataAdapter(SQL,myConnection);
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
    DataTable myDataTable = new DataTable();
    myAdapter.Fill(myDataTable);

    ListeCommandes.DataSource =  myDataTable.DefaultView;
    ListeCommandes.DataBind();

    if(ListeCommandes.Items.Count==0)
        ListeVide.Visible = true;
    else
        ListeVide.Visible = false;
}
```

- Récupération de la connexion partagée, stockée dans l'objet `Session`
- Définition de la source de données (procédure stockée `ListeCommandes`). On utilise arbitrairement la méthode `SqlDataAdapter`/`DataTable` (il aurait été possible d'utiliser `SqlCommand`/`SqlDataReader` à la place)
- Établissement du lien entre le contrôle `Repeater` et la source de données
- Affichage d'un message d'information « la liste des commandes est vide » au cas où la liste est vide

Voici le code correspondant en VB.NET :

Fournisseurs.aspx (Version VB.NET)

```
Sub Page_Load(sender As Object,e As EventArgs)

    Dim myConnection As SqlConnection
    Dim myAdapter As SqlDataAdapter
    Dim myDataTable As DataTable
    Dim Sql As String

    myConnection = CType(Session("myConnection"),SqlConnection)

    SQL = "ListeCommandes"

    myAdapter = new SqlDataAdapter(SQL,myConnection)
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure

    myDataTable = new DataTable()
    myAdapter.Fill(myDataTable)

    ListeCommandes.DataSource =  myDataTable.DefaultView
    ListeCommandes.DataBind()
```

```
    if (ListeCommandes.Items.Count=0)
        ListeVide.Visible = True
    Else
        ListeVide.Visible = False
    End If

End Sub
```

La liaison avec la source de données étant effectuée, il faut maintenant spécifier comment afficher les éléments de cette source de données.

Réalisation de la maquette HTML des éléments du contrôle Repeater

La syntaxe générale de mise en œuvre d'un contrôle `Repeater` est la suivante (seuls les éléments en couleur sont obligatoires) :

```
<asp:Repeater id="MyRepeater" runat="server">
   <HeaderTemplate> (facultatif)
   Contenu HTML correspondant à l'en-tête
   </HeaderTemplate>
   <ItemTemplate>  (obligatoire)
   Contenu HTML correspondant aux éléments
   </ItemTemplate>
   <AlternatingItemTemplate> (facultatif)
   Contenu HTML correspond aux éléments pairs
   </AlternatingItemTemplate>
   <SeparatorTemplate> (facultatif)
   Contenu HTML correspond au séparateur
  </SeparatorTemplate>
   <FooterTemplate>
   Contenu HTML correspondant au pied
   </FooterTemplate>
</asp:Repeater>
```

À propos d'AlternatingItemTemplate

La section `<AlternatingItemTemplate>` permet de définir, si on le souhaite, une maquette alternative à utiliser pour l'affichage d'un élément sur deux : ceci peut, par exemple, permettre de rendre la liste plus lisible.

DataGrid, Repeater et DataList : trois contrôles cousins

Il existe trois contrôles serveur permettant d'afficher une liste de données :

- le contrôle `DataGrid`, qui permet d'afficher une grille de données extraites d'une source, est le plus simple à utiliser mais aussi le moins souple (il s'affiche toujours sous la forme d'un tableau HTML) en dépit des nombreuses possibilités de formatage et de paramétrage des colonnes (voir chapitre précédent), ainsi que du support intégré de la pagination ;
- le contrôle `Repeater`, qui permet d'afficher une liste de données en spécifiant la maquette HTML de chaque élément, est moins simple à utiliser (saisie du code HTML obligatoire) mais plus souple si on souhaite réaliser des affichages spécifiques ;
- le contrôle `DataList` est une version améliorée de `Repeater`, qui, outre l'affichage d'une liste d'éléments suivant une maquette HTML définie par l'utilisateur, permet également la sélection et l'édition d'éléments, ainsi que l'affichage sur plusieurs colonnes.

Parmi ces trois contrôles, seul `DataGrid` dispose du support intégré de la pagination : cependant, la classe `PagedDataSource` permet d'implémenter rapidement un mécanisme de pagination pour un contrôle `Repeater` ou `DataList`, comme nous le signalons plus loin.

Dans notre cas, nous allons utiliser l'élément `ItemTemplate` pour afficher le détail de la commande, c'est-à-dire (voir figure 5-4) :

- le numéro de la commande ;
- le nom du fournisseur ;
- la date de livraison prévue ;
- le statut de la livraison ;
- un bouton permettant d'accéder au détail de la commande ;
- un bouton permettant de supprimer la commande.

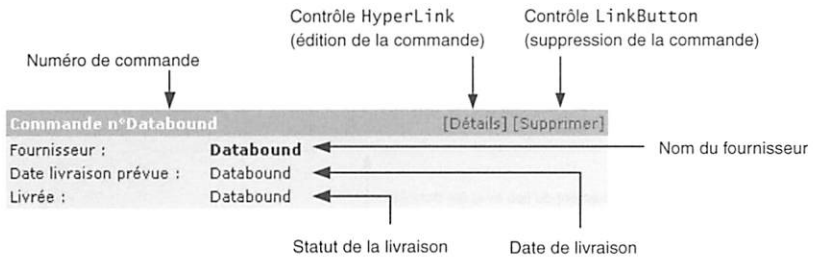


Figure 5–4 Maquette de l'élément `Item` du `Repeater`

Le point principal d'architecture est la réalisation de la liaison entre les éléments de la maquette et les valeurs correspondantes de la source de données :

- Lorsque la méthode `DataBind()` du contrôle `Repeater` est appelée, tous les enregistrements de la source de données sont parcourus séquentiellement.
- À chaque enregistrement est associé dynamiquement un contrôle nommé `Container`, qui correspond à l'élément graphique `ItemTemplate` du `Repeater` pour cet enregistrement ; ce contrôle a une propriété nommée `DataItem`, de type `DataRowView` qui contient les valeurs de l'enregistrement.

Plus d'informations sur DataBinder

`DataBinder` est une classe utilitaire qui permet de simplifier la syntaxe des expressions de liaisons de données.
Sans cette classe, nous aurions du écrire, dans le cas d'une page en C# :

```
<%# ((DataRowView)Container.DataItem)
➥ ["DateLivraison"] %>
```

Avec `DataBinder`, il n'est pas nécessaire d'effectuer explicitement la conversion de type :

```
<%# DataBinder.Eval(Container.DataItem,
➥ "DateLivraison") %>
```

L'avantage de cette classe utilitaire réside surtout dans la gestion du formatage des données affichées. Ainsi, au lieu d'écrire :

```
<%# String.Format("{0:c}",
➥ ((DataRowView)Container.DataItem)
➥ ["DateLivraison"]) %>
```

on peut écrire :

```
<%# DataBinder.Eval(Container.DataItem,
➥ "DateLivraison","{0:d}") %>
```

ATTENTION

<%# ... %> n'a rien à voir avec <%...%>

Dans la syntaxe ASP.NET, une directive de type <%# ... %> désigne une instruction de liaison de données qui est compilée, puis évaluée dynamiquement lors de l'appel de la méthode `DataBind` pour la page ou les contrôles concernés, alors que <%...%> désigne un script ASP qui est interprété lors de l'exécution de la page.

- Le lien entre le contenu graphique et les champs de `Container.DataItem` s'effectue grâce à une syntaxe spéciale dite de « liaison de données » (*data-binding*), insérée dans la partie graphique de la page, qui permet de récupérer dynamiquement la valeur du champ <NomDuChamp> pour l'élément en cours de génération :

```
<%# DataBinder.Eval(Container.DataItem,"<NomDuChamp>") %>
```

- Après évaluation dynamique de toutes les instructions de liaison de données, l'instanciation de la maquette HTML correspondant à l'enregistrement est générée (voir figure 5-5).

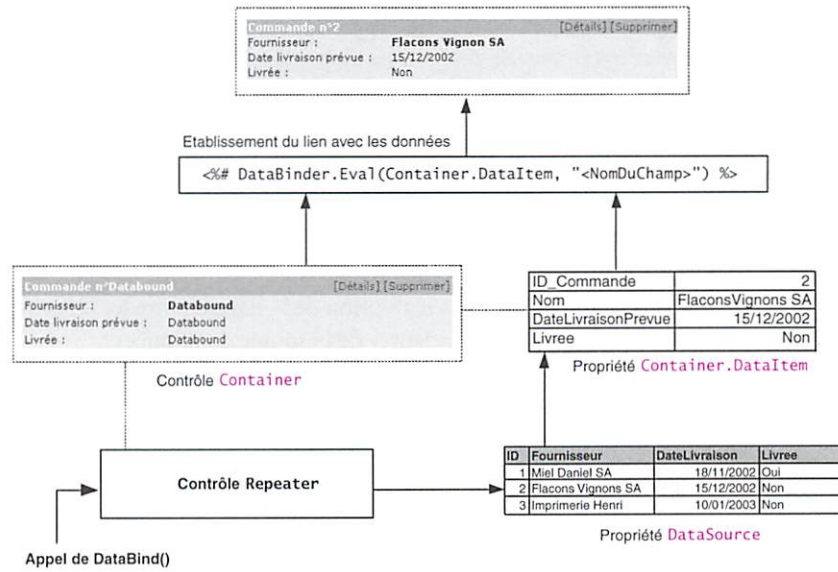


Figure 5–5 Mécanisme de lien aux données pour le contrôle Repeater

Nous allons utiliser cette syntaxe de liaison de données pour personnaliser notre contrôle `Repeater` afin qu'il affiche, pour chaque élément, un tableau HTML composé de quatre lignes : la première spécifie le numéro de la commande et contient les boutons Détails (contrôle `HyperLink`) et Supprimer (contrôle `LinkButton`), et les trois suivantes spécifient respectivement le nom du fournisseur, la date de livraison prévue et le statut de la livraison :

Fournisseurs.aspx (partie graphique/contrôle Repeater)

La section <ItemTemplate> contient le code HTML à répéter

```
<asp:Repeater id="ListeCommandes" runat="server">
  <ItemTemplate>
    <table bordercolor="black" cellspacing="0">
      <tr bgcolor="#aaaadd">
        <td width="150" >
          <b><font color="white"> Commande n°
```

```
      <%# DataBinder.Eval(Container.DataItem, "ID_Commande") %>
     </font></b>
        </td>
        <td width="300" align="right">
          <asp:HyperLink runat="server">[Détails]</asp:HyperLink>
          <asp:LinkButton runat="server">[Supprimer]</asp:LinkButton>
        </td>
      </tr>
      <tr bgcolor="#ffffc0">
        <td width="150">Fournisseur : </td>
        <td width="300">
          <b><%# DataBinder.Eval(Container.DataItem, "NomFournisseur") %>
          </b>
          </td>
      </tr>
      <tr bgcolor="#ffffc0">
         <td width="150">Date livraison prévue : </td>
         <td width="300">
          <%# DataBinder.Eval(Container.DataItem,"DateLivraison")%>
         </td>
      </tr>
      <tr bgcolor="#ffffc0">
        <td width="150">Livrée : </td>
        <td width="300">
          <%# DataBinder.Eval(Container.DataItem,"Livree") %>
       </td>
      </tr>
    </table>
  </ItemTemplate>
  <SeparatorTemplate>
  <br />
  </SeparatorTemplate>
</asp:Repeater>
```

Retrouve le numéro de la commande courante

Pour l'instant, les boutons Détails et Supprimer sont inactifs (ils seront implémentés par la suite)

Retrouve le nom du fournisseur correspondant à la commande courante

Retrouve la date de livraison de la commande courante

Retrouve le statut de la livraison de la commande courante

La section <SeparatorTemplate> contient le code HTML à insérer entre deux éléments (on utilise un simple saut de ligne)

Le résultat de l'exécution de la page (figure 5-6) est déjà appréciable, mais non encore parfait : le format d'affichage de la date laisse à désirer, le statut de livraison est affiché en anglais et les boutons Détails et Supprimer sont, pour l'instant, inactifs. Nous allons remédier à cela, point par point, dans les paragraphes qui suivent.

Formatage des éléments liés aux données

Grâce aux expressions de liaison de données, nous pouvons donc personnaliser facilement les éléments d'un contrôle `Repeater` en fonction du contenu de la source de données associée. Néanmoins, en l'absence d'indications supplémentaires, c'est l'affichage par défaut qui est utilisé, par exemple pour une date :

```
01/12/2002 00:00:00
```

Figure 5–6
Affichage de la liste des commandes

Si on souhaite contrôler plus précisément l'affichage des données (par exemple masquer l'heure dans l'affichage d'une date), il faut avoir recours aux fonctionnalités de la méthode `Format` de la classe `String`, dont la variante la plus utilisée est :

```
public static String Format (String format, Object arg0)
```

où :

- `arg0` spécifie l'objet dont on souhaite afficher la valeur ;
- `format` désigne la chaîne spécifiant le format à appliquer à cet objet.

Les chaînes de formatage dans les applications .NET

Une chaîne de formatage permet de spécifier le format d'affichage d'une variable.

La syntaxe générale d'une chaîne de formatage est :

```
{ N [, M ][: formatString ]}
```

où :

- `N` est une valeur entière positive ou nulle qui spécifie l'indice de l'argument concerné par le formatage (0 dans le cas où il n'y a qu'un seul argument) ;
- `M` est une valeur entière strictement positive ou négative qui désigne la largeur de la zone à utiliser pour l'affichage de la valeur formatée, laquelle est alors complétée par autant d'espaces que nécessaire (justification à gauche si `M` est positif, à droite si `M` est négatif) ; cet argument est optionnel ;
- `formatString` est une chaîne de formatage standard (réduite à un caractère et liée aux paramètres d'affichage de l'application) ou spécifique (spécifiant précisément l'affichage à réaliser) ; cet argument est optionnel.

À NOTER Fonctions de formatage utilisables dans toutes les applications .NET

Les fonctionnalités liées au formatage sont utilisables non seulement dans les applications Web réalisées avec ASP.NET, mais plus généralement dans toute application .NET.

On ne décrira pas ici l'ensemble des options de formatage possibles (le lecteur est invité à se reporter à la documentation MSDN à l'adresse http://msdn.microsoft.com/library). Néanmoins, à titre d'exemples, on présente quelques-unes des options disponibles pour le formatage des dates.

Les formats standards sont liés à la langue

Les formats standards dépendent de la langue de l'application (par exemple, « d » donnera « 15/11/2002 » en français et « 11/15/2002 » en anglais). Égale par défaut à la langue du système d'exploitation, la langue de l'application peut être changée grâce au fichier `web.config` (voir chapitre 9).

Tableau 5–1 Quelques exemples de formats applicables à une date

Chaîne	Type de chaîne	Résultat	Description
d	Standard	15/11/2002	Date format court
D	Standard	vendredi 15 novembre 2002	Date format long
t	Standard	17:57	Heure format court
T	Standard	17:57:53	Heure format long
g	Standard	15/11/2002 17:58	Date et heure format court
dd-MM-yy	Spécifique	15-11-02	Format spécifique

Ainsi, pour spécifier l'affichage de la date de livraison sous la forme courte dans la liste des commandes, il suffit d'ajouter la chaîne de formatage `{0:d}` en paramètre de la méthode `Eval` de `DataBinder` :

Fichier Fournisseurs.aspx (Partie graphique – Extrait)

```
<asp:Repeater ...>
...
    <%# DataBinder.Eval(Container.DataItem,"DateLivraison","{0:d}") %>
...
</asp:Repeater>
```

Les chaînes de formatage constituent un mécanisme puissant mais non universel : notamment, elles ne gèrent pas la personnalisation de l'affichage des champs booléens en fonction de la langue.

Pour afficher le statut de livraison de la commande sous la forme Oui/Non plutôt que True/False, nous allons devoir développer une fonction spécifique, `AfficherBooleen`, à placer dans le code de la page `Fournisseurs.aspx`.

Fichier Fournisseurs.aspx (Fonction AfficherBooleen/Version C#)

```
string AfficherBooleen(object o)
{
    return ( (bool)o ? "Oui" : "Non");
}
```

Fichier Fournisseurs.aspx (Fonction AfficherBooleen/Version VB.NET)

```
Function AfficherBooleen(o As Object) As String

    If (o=True) Then
        return "Oui"
    Else
        return "Non"
    End If

End Function
```

Cette fonction pourra ensuite être appelée depuis l'instruction de liaison de données située dans la partie graphique de la page :

Fichier Fournisseurs.aspx (Partie graphique – Extrait)

```
<asp:Repeater ...>
...
    <%# AfficherBooleen(DataBinder.Eval(Container.DataItem,"Livree")) %>
...
</asp:Repeater>
```

Le formatage des données affichées étant réalisé, passons maintenant à l'implémentation du bouton Détails, qui doit permettre d'accéder au détail d'une commande.

Paramétrage du bouton Détails pour accéder au détail d'une commande

Le bouton Détails associé à une commande doit permettre d'accéder au détail de cette commande, qui sera affiché grâce à la page `DetailCommande.aspx` à laquelle on passera en paramètre le numéro de la commande concernée.

Pour réaliser cette liaison, nous allons simplement paramétrer la propriété `NavigateUrl` du contrôle `HyperLink` associé au bouton Détails, en utilisant les fonctionnalités de formatage de la classe `DataBinder` :

Fichier Fournisseurs.aspx (Partie graphique – Extrait)

```
<asp:Repeater ...>
...
<asp:HyperLink runat="server"
   NavigateUrl=<%#DataBinder.Eval(Container.DataItem,
      ➥ "ID_Commande","DetailCommande.aspx?id={0}")%> >
[Détails]</asp:HyperLink>
...
</asp:Repeater>
```

La figure 5-7 représente la nouvelle version de la liste des commandes après les améliorations apportées au formatage et le paramétrage du bouton Détails ; passons pour finir à l'implémentation du bouton Supprimer, pour l'instant toujours inactif.

Figure 5–7
La liste des commandes après formatage

Gestion de la suppression d'une commande avec LinkButton

On souhaite que, lorsque l'utilisateur clique sur le bouton Supprimer associé à une commande, cette commande soit supprimée de la base de données et que la liste soit rechargée ; idéalement, il faudrait afficher un message de confirmation avant d'effectuer la suppression effective, pour donner une chance à l'utilisateur de changer d'avis.

Nous allons implémenter cette cinématique en utilisant la gestion des événements et un contrôle `LinkButton`, qui a l'apparence d'un lien hypertexte mais se comporte comme un bouton, dans le sens où il génère un événement lorsqu'on clique dessus (voir figure 5-8).

Si le mécanisme de gestion des événements est similaire à celui que nous avons détaillé à la fin du chapitre précédent (postage automatique de la page, remontée d'une nouvelle instance avec exécution du gestionnaire d'événement concerné), une difficulté supplémentaire est introduite du fait que le contrôle source de l'événement (`LinkButton`) est à l'intérieur d'un autre contrôle (`Repeater`).

À RETENIR Différence entre HyperLink et LinkButton

Le contrôle serveur `HyperLink` implémente un lien hypertexte classique, tandis que `LinkButton` implémente un bouton qui a l'apparence d'un lien mais qui provoque la génération d'un événement lorsqu'on clique dessus.

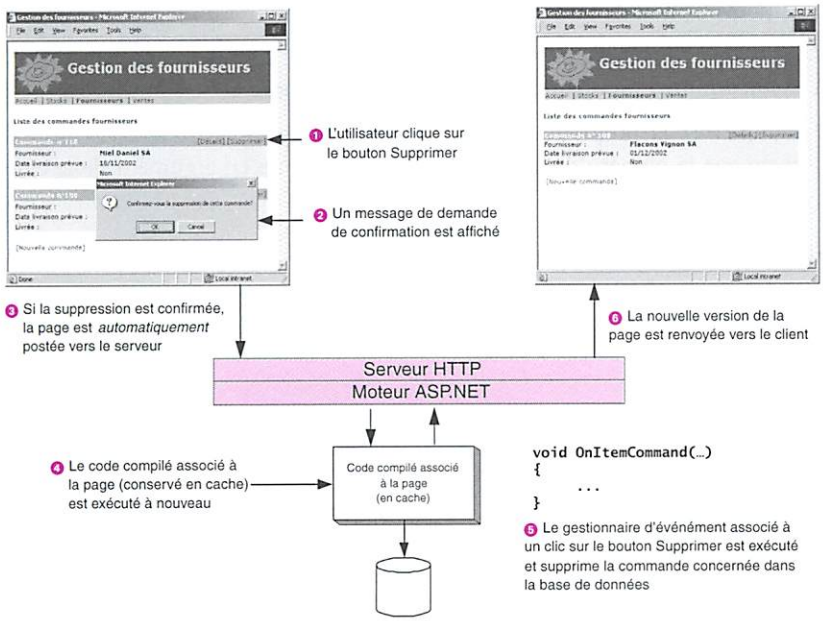


Figure 5–8 Cinématique de la suppression d'une commande

En effet, un contrôle situé à l'intérieur d'un autre contrôle ne génère pas directement un événement au niveau de la page : l'événement est géré par le contrôle conteneur puis passé à la page sous la forme d'un événement `OnItemCommand` (voir figure 5-9).

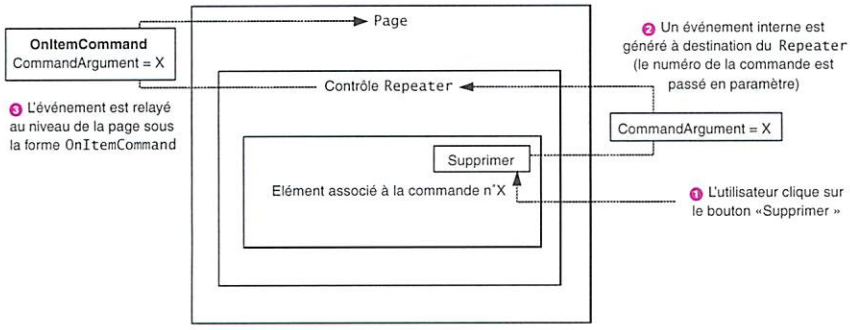


Figure 5–9 Mécanisme de remontée des événements depuis un contrôle interne

Comme plusieurs boutons **Supprimer** peuvent être situés à l'intérieur du même contrôle `Repeater`, il faut pouvoir identifier lequel d'entre eux est à l'origine de l'événement `OnItemCommand` : ceci est possible grâce à la propriété `CommandArgument`, qui permet de remonter des informations associées à la source de l'événement (en l'occurrence, le numéro de la commande).

Remontée d'événements en bulle (event bubbling)

Ce mécanisme de remontée des événements depuis un contrôle interne est connu sous le nom de « remontée d'événements en bulle » (*event bubbling*), du fait que l'événement remonte progressivement vers les objets de plus haut niveau, à l'image de bulles de champagne remontant vers la surface.

Nous pouvons maintenant procéder à l'implémentation du mécanisme de suppression des commandes. Commençons par la partie graphique de la page :

1. Associez un gestionnaire à l'événement `OnItemCommand` du contrôle `Repeater`.
2. Paramétrez la propriété `CommandArgument` du contrôle `LinkButton` de telle sorte que le numéro de la commande soit remonté en argument de la commande.

Fichier Fournisseurs.aspx (Partie graphique – Extrait)

```
<asp:Repeater id="ListeCommandes" runat="server"
   OnItemCommand="OnItemCommand" >
...
<asp:LinkButton
   CommandArgument='<%# DataBinder.Eval(Container.DataItem,
   ➥ "ID_Commande") %>'
   runat="server">[Supprimer]</asp:LinkButton>
   ...
</asp:Repeater>
```

Ceci assure que l'événement `OnItemCommand` sera déclenché lorsque l'utilisateur cliquera sur un bouton Supprimer, le numéro de la commande concernée étant passé en paramètre du gestionnaire d'événement.

RAPPEL Ordre d'appel des gestionnaires d'événements

Les gestionnaires d'événements déclenchés par les contrôles (en l'occurrence `OnItemCommand`) sont appelés après les gestionnaires d'initialisation de la page (`Page_Init` puis `Page_Load`).

Implémentons maintenant le gestionnaire `OnItemCommand` qui réalise la suppression de la commande dans la base. Cet événement étant appelé après `Page_Load`, il faut déplacer le chargement de la liste, afin qu'il ait lieu une fois la suppression effectuée (faute de quoi, l'élément supprimé apparaîtra encore dans la liste) ; pour cela, nous allons utiliser une fonction utilitaire `ChargerListeCommandes` et partager la connexion au niveau de la page.

Fichier Fournisseurs.aspx (Version C#)

```
SqlConnection   myConnection;

void Page_Load(Object sender, EventArgs e)
{

    myConnection = (SqlConnection)Session["myConnection"];

    if(!IsPostBack)
    {
        ChargerListeCommandes();
    }
}

void ChargerListeCommandes()
{
    string SQL = "ListeCommandes";

    SqlDataAdapter myAdapter = new SqlDataAdapter(SQL,myConnection);
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
    DataTable myDataTable = new DataTable();
    myAdapter.Fill(myDataTable);

    ListeCommandes.DataSource =  myDataTable.DefaultView;
    ListeCommandes.DataBind();
```

On implémente la connexion sous la forme d'une variable partagée au sein de la page, afin de pouvoir y faire référence dans toutes les fonctions.

Initialisation de la connexion.

On ne charge la liste des commandes que lors du premier affichage de la page (si la page est repostée suite à une demande de suppression, la liste sera chargée depuis `OnItemCommand`, après suppression de la commande).

Cette fonction charge la liste des commandes (code inchangé par rapport à l'ancienne version de `Page_Load` développée au début de ce chapitre).

```
    ListeVide.Visible = (ListeCommandes.Items.Count==0 ? true : false);
}

void OnItemCommand(Object sender, RepeaterCommandEventArgs e)
{
    string SQL = "DELETE FROM Commande WHERE ID_Commande = "
    ➥  +e.CommandArgument;
    SqlCommand myCommand = new SqlCommand(SQL,myConnection);
    myCommand.ExecuteNonQuery();

    ChargerListeCommandes();
}
```

Appelé lorsque la page est repostée suite à un clic sur un bouton Supprimer, ce gestionnaire d'événement supprime dans la base la commande dont le numéro est contenu dans `e.CommandArgument` puis charge la liste des commandes.

La version VB.NET de ce code, avec commentaires, est disponible en ligne à l'adresse http://www.savonsdusoleil.com/src/vb/Fournisseurs.aspx.

Il ne manque plus qu'un élément pour parfaire notre mécanisme de suppression des commandes : l'affichage d'un message de confirmation lorsque l'utilisateur clique sur le bouton Supprimer.

Utilisation de l'événement OnItemCreated pour ajouter un message de confirmation

Pour afficher un message de confirmation de la suppression côté client, il faut ajouter un script associé à l'événement client `onclick`, de manière à ce que le code HTML généré soit le suivant :

```
<a href="..."
 onclick="return confirm('Confirmez-vous la suppression de cette
  ➥ commande?');" >
Supprimer</a>
```

Malheureusement, si on saisit directement ce script dans le contenu graphique de la page, il sera interprété comme un événement serveur (en effet, `OnClick` est un nom réservé pour le gestionnaire associé à l'événement `Click` du contrôle `LinkButton`).

La solution est donc d'implémenter l'ajout dynamique de cet attribut client `onclick` depuis le code de la page, en utilisant pour cela l'événement `OnItemCreated`, appelé chaque fois qu'un élément du contrôle `Repeater` est créé.

Commençons par modifier la partie graphique de la page :

1. Associez un gestionnaire à l'événement `OnItemCreated` du contrôle `Repeater`.
2. Paramétrez la propriété `id` du contrôle `LinkButton` afin de pouvoir identifier le contrôle.

Fichier Fournisseurs.aspx (Partie graphique - Extrait)

```
<asp:Repeater id="ListeCommandes" runat="server"
    OnItemCommand="OnItemCommand"
    OnItemCreated="OnItemCreated" >
...
```

Ceci assure que l'événement `OnItemCreated` sera appelé lors de la création de chaque élément du `Repeater`.

L'identification du contrôle est indispensable pour pouvoir utiliser la méthode `FindControl`.

```
<asp:LinkButton id="btnDelete"
   CommandArgument='<%# DataBinder.Eval(Container.DataItem,
      ➥ "ID_Commande") %>'
   runat="server">[Supprimer]</asp:LinkButton>
...
</asp:Repeater>
```

Ceci étant fait, il ne reste plus qu'à implémenter la gestion `OnItemCreated` dans le code de la page :

Fichier Fournisseurs.aspx (Gestionnaire OnItemCreated – Version C#)

Si l'appel de ce gestionnaire correspond à la création d'un élément de type `Item`, on recherche le bouton Supprimer grâce à la méthode `FindControl`, puis on ajoute un attribut HTML affichant un message de confirmation grâce à la collection `Attributes`.

```
void OnItemCreated(Object sender, RepeaterItemEventArgs e)
{
  LinkButton myDeleteButton;
  if (e.Item.ItemType==ListItemType.Item)
  {
    myDeleteButton = (LinkButton)e.Item.FindControl("btnDelete");
    myDeleteButton.Attributes.Add("onclick",
      ➥ "return confirm('Confirmez-vous la suppression de cette
      ➥  commande?');");
  }
}
```

Fichier Fournisseurs.aspx (Gestionnaire OnItemCreated – Version VB.NET)

```
Sub OnItemCreated(sender As Object,e As RepeaterItemEventArgs)
  Dim myDeleteButton As LinkButton
  If  (e.Item.ItemType=ListItemType.Item) Then
       myDeleteButton = e.Item.FindControl("btnDelete")
       myDeleteButton.Attributes.Add("onclick",
         ➥ "return confirm('Confirmez-vous la suppression de cette
         ➥  commande?');")
  End If
End Sub
```

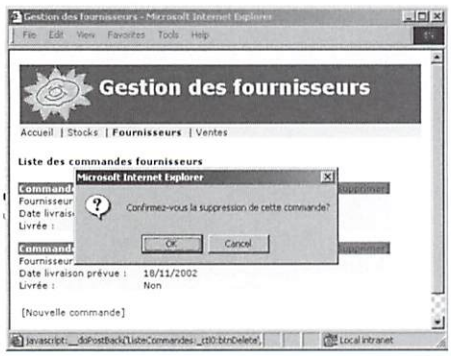


Figure 5–10 Confirmation de la suppression d'une commande

Testez le bon fonctionnement de la page : un message de confirmation doit apparaître lorsque vous cliquez sur un bouton Supprimer (voir figure 5-10).

Édition d'une commande existante

L'affichage de la liste des commandes, avec gestion de la suppression, étant implémenté, passons au mécanisme d'édition des commandes : l'utilisateur doit pouvoir, en cliquant sur le bouton Détails associé à une commande, consulter le détail de la commande (en-tête et lignes de commande) et modifier, s'il le souhaite, le statut de la livraison (la commande est-elle livrée : « oui » ou « non » ?)

Nous allons implémenter ce mécanisme grâce à la page `DetailCommandes.aspx`, principalement basée sur l'utilisation de la classe `SqlDataAdapter`, dont nous allons commencer par réaliser la maquette.

Aller plus loin : ajouter un mécanisme de pagination à la liste des commandes

Contrairement à son cousin `DataGrid`, le contrôle `Repeater` ne dispose pas de fonctionnalités intégrées pour la gestion de la pagination (affichage d'un nombre limité de commandes par page et boutons permettant de naviguer entre les pages).

Néanmoins, il est relativement facile de mettre en place un mécanisme de pagination en associant au contrôle `ListeCommandes` une source de données de type `PagedDataSource`, elle-même liée à la procédure `ListeCommandes` (voir figure 5-11).

La classe `PagedDataSource` gère tous les détails nécessaires à la visualisation d'une source de données de manière paginée ; il suffit de spécifier la taille de la page et l'index de la page courante :

```
PagedDataSource myPager;
myPager = new PagedDataSource();
myPager.DataSource =  myDataTable.DefaultView;
myPager.AllowPaging = true;
myPager.PageSize    = 3;
myPager.CurrentPageIndex = index;
ListeCommandes.DataSource = myPager;
```

DataGrid utilise PagedDataSource

Le contrôle `DataGrid` utilise de manière interne la classe `PagedDataSource` pour sa gestion intégrée de la pagination ; heureusement, cette classe peut également être utilisée directement, comme nous le faisons ici.

Pour compléter l'implémentation, il faut rajouter deux boutons Page Précédente et Page Suivante en bas de la liste, qui permettent respectivement de décrémenter ou d'incrémenter l'index de la page courante, en utilisant, par exemple, le mécanisme de gestion des événements et la propriété `ViewState` de la classe `Page` pour conserver la valeur de l'index entre deux chargements de la page (cela évite l'utilisation fastidieuse de contrôles cachés ou d'arguments sur la chaîne de requête, ainsi que le recours à l'objet `Session`, consommateur de ressources sur le serveur).

Code source complet téléchargeable en ligne

Dans cette section, nous ne faisons que donner des indications générales pour l'implémentation de la fonctionnalité de pagination ; le code source complet (VB.NET et C#) est disponible sur le site :

- www.savonsdusoleil.com

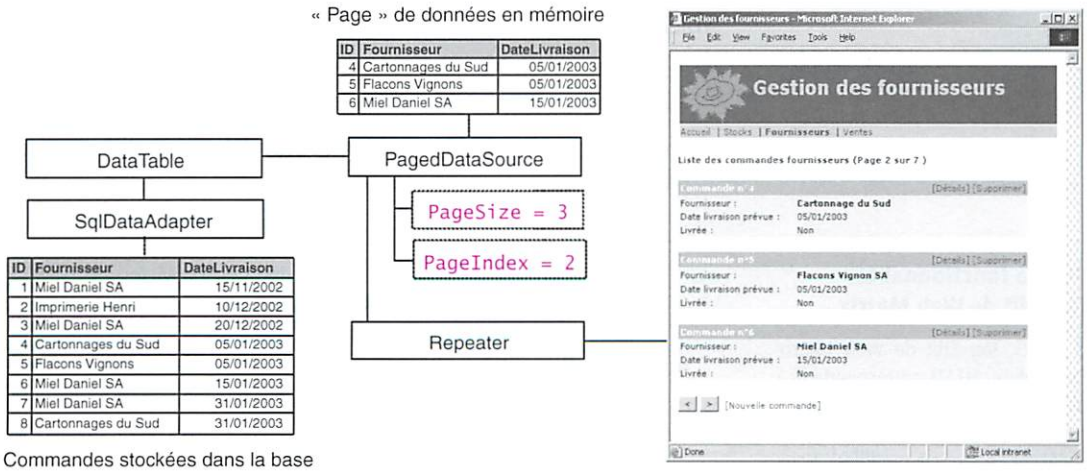


Figure 5–11 Mise en place d'une source de données paginée

Réalisation de la maquette de la page affichant le détail d'une commande

La page de consultation du détail d'une commande doit afficher le numéro de la commande, le nom du fournisseur, la date de livraison prévue, le statut de la livraison et le détail des lignes de commandes ; toutes ces informations doivent être en lecture seule, à l'exclusion du statut de la livraison qui doit être modifiable par l'utilisateur (voir figure 5-12).

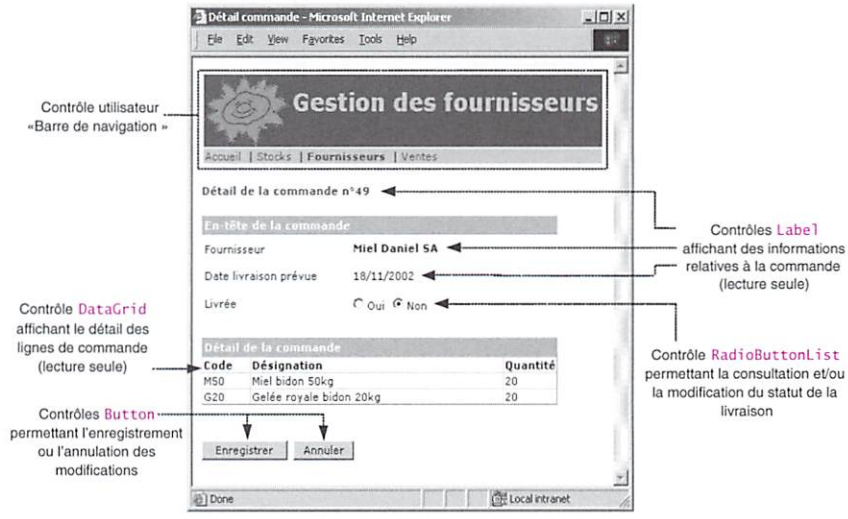


Figure 5–12 La page de consultation du détail d'une commande

Voici la marche à suivre pour la réalisation de la maquette :

1. Démarrez Web Matrix et ouvrez le fichier `DetailCommande.aspx` créé précédemment.
2. Saisissez un titre (« Détail de la commande n° ») suivi d'un contrôle serveur de type `Label` nommé `NoCommande`.

ALTERNATIVE La fonctionnalité Quick Tag Edit de Web Matrix

La fonctionnalité Quick Tag Edit de Web Matrix permet d'éditer le contenu HTML correspondant à un élément graphique isolé (évitant ainsi au développeur de basculer dans l'onglet HTML et de perdre du temps à localiser l'endroit à modifier). Vous pouvez, par exemple, spécifier les caractéristiques du contrôle `StatutLivraison` en le sélectionnant dans l'onglet Design et en choisissant Edit Tag dans le menu Edit (ou dans le menu contextuel) ; voir figure 5-13.

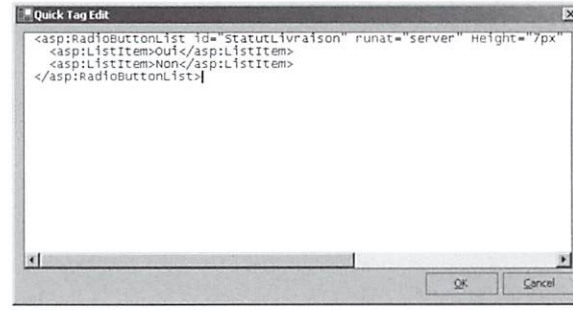


Figure 5–13 La fonctionnalité Quick Tag Edit de Web Matrix

3 Créez un tableau HTML composé de quatre lignes : une ligne de titre (« En-tête de la commande ») comportant une seule colonne et trois lignes comportant deux colonnes affichant respectivement :
   - le nom du fournisseur (contrôle serveur `NomFournisseur` de type `Label`) ;
   - la date de livraison prévue (contrôle serveur `DateLivraison` de type `Label`) ;
   - le statut de la livraison (contrôle serveur `StatutLivraison` de type `RadioButtonList`).

4 En basculant dans l'onglet HTML, spécifiez les deux valeurs possibles pour le contrôle `StatutLivraison` (vous pouvez également utiliser la fonctionnalité Quick Tag Edit de Web Matrix).

```
<asp:RadioButtonList id="StatutLivraison" runat="server" ">
<asp:ListItem>Oui</asp:ListItem>
<asp:ListItem>Non</asp:ListItem>
</asp:RadioButtonList>
```

5 Créez un second tableau HTML composé d'une seule ligne et d'une seule colonne, contenant le texte « Lignes de commandes ».

6 Immédiatement en dessous, insérez un contrôle serveur de type `DataGrid` nommé `LignesCommande`.

7 À l'aide de l'assistant Property Builder de Web Matrix (déjà utilisé au chapitre précédent), désactivez l'option Create columns automatically at run time pour ce contrôle et ajoutez trois colonnes de type `Bound Columns` ayant les caractéristiques suivantes :

Header Text	Data Field
Code	Code
Désignation	Designation
Quantité	Quantite

8 Ajoutez deux contrôles serveur de type `Button` respectivement nommés Enregistrer et Annuler.

Le résultat doit ressembler à celui présenté à la figure 5-14.

Afin de pouvoir réaliser le traitement adéquat lorsque l'utilisateur cliquera sur les boutons Enregistrer ou Annuler, déclarons deux gestionnaires d'événements :

Contrôle	Événement	Gestionnaire
Enregistrer	`Click`	`OnEnregistrerClicked`
Annuler	`Click`	`OnAnnulerClicked`

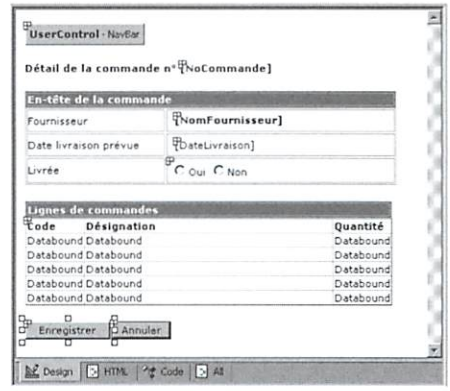


Figure 5–14 Maquette de la page Détails de la commande

Déclarer un gestionnaire d'événement avec Web Matrix

En présentant le mécanisme de gestion des événements à la fin du chapitre précédent, nous avions réalisé toute la saisie du code manuellement ; pour aller plus vite, on peut également utiliser les fonctionnalités de l'interface graphique de Web Matrix.

Pour déclarer un gestionnaire d'événement avec Web Matrix :

1. Sélectionnez le contrôle à la source de l'événement.
2. Dans la feuille de propriétés associée au contrôle, cliquez sur l'icône Events : la liste des événements associés au contrôle apparaît.
3. Saisissez le nom du gestionnaire à implémenter en face de l'événement choisi (voir figure 5-15 : gestionnaire `OnEnregistrer Clicked` associé à l'événement `Click` du bouton).
4. Appuyez sur la touche Entrée pour valider la saisie.

Cette manipulation effectue deux ajouts :

- insertion de l'appel du gestionnaire d'événement dans le contenu HTML ;
- implémentation du squelette du gestionnaire dans le code.

ASTUCE Utiliser le nom par défaut

Si on effectue un double-clic sur la zone de saisie du nom, un gestionnaire d'événement portant un nom par défaut (`NomControle_NomEvenement`) est généré.

Éléments ajoutés dans la partie graphique :

```
<asp:Button id="Enregistrer"
 onclick="OnEnregistrerClicked" runat="server" />
```

Éléments ajoutés dans la partie code (ici en C#) :

```
void OnEnregistrerClicked(Object sender, EventArgs e)
{
...
}
```

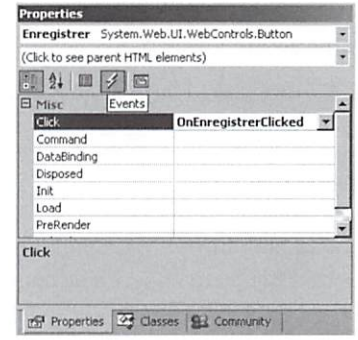


Figure 5–15 La feuille de propriétés Événements

Voici le contenu HTML correspondant (les contrôles serveur sont indiqués en couleur) :

DetailCommande.aspx (partie graphique)

```
...
<form runat="server">
  <h4>Détail de la commande n°
<asp:Label id="NoCommande" runat="server"/></h4>
  <table bordercolor="black">
   <tr bgcolor="#aaaadd">
      <td width="100%" colspan="2">
        <font color="white"><b>En-tête de la commande</b></font></td>
   </tr>
   <tr valign="center" height="25">
      <td width="150">Fournisseur </td>
      <td width="250">
     <asp:Label id="NomFournisseur" runat="server"/></td>
   </tr>
   <tr valign="center" height="25">
      <td width="150">Date livraison prévue</td>
      <td width="250">
      <asp:Label id="DateLivraison" runat="server"/> </td>
   </tr>
```

```
  <tr valign="center" height="25">
     <td width="150">Livrée</td>
     <td width="250">
       <asp:RadioButtonListid="StatutLivraison" runat="server">
         <asp:ListItem>Oui</asp:ListItem>
         <asp:ListItem>Non</asp:ListItem>
       </asp:RadioButtonList>
    </td></tr>
 </table>
 <br>
 <table bordercolor="black" bgcolor="#aaaadd">
  <tr>
    <td width="380">
    <font color="white"><b>Lignes de commandes</b></font></td>
  </tr>
 </table>
 <asp:DataGrid id="LignesCommande" runat="server"
      AutoGenerateColumns="false">
  <Columns>
     <asp:BoundColumn HeaderText="Code" DataField="Code" />
    <asp:BoundColumn HeaderText="Désignation" DataField="Designation" />
     <asp:BoundColumn HeaderText="Quantité" DataField="Quantite" />
 </asp:DataGrid>
 <br>
 <asp:Button id="Enregistrer" runat="server"
      Text="Enregistrer" onclick="OnEnregistrerClicked"/>
 <asp:Button id="Annuler" runat="server"
 Text="Annuler"  onclick="OnAnnulerClicked" />
</form>
...
```

La maquette étant terminée, passons maintenant à l'implémentation du code de la page.

Consultation et mise à jour de la commande avec SqlDataAdapter

La cinématique à implémenter peut être résumée de la manière suivante :

- au chargement de la page, mise à jour des valeurs des contrôles serveur à partir des informations contenues dans la base de données ;
- en cas de clic sur le bouton Enregistrer, mise à jour de la base de données en fonction des modifications effectuées puis retour à la liste des commandes ;
- en cas de clic sur le bouton Annuler, retour à la liste des commandes sans faire de modifications à la base de données.

La pièce maîtresse de l'implémentation sera la classe `SqlDataAdapter`, qui permet de gérer la synchronisation entre la base de données et un objet `DataTable`, tant dans le sens de la lecture (chargement des données) que dans le sens de l'écriture (mise à jour des données), comme l'illustre la figure 5-16.

La classe SqlDataAdapter

La classe `SqlDataAdapter` gère la synchronisation entre une base de données SQL et un objet de type `DataTable` ou `DataSet` situé en mémoire ; elle comporte quatre propriétés de type `SqlCommand` qui permettent à l'utilisateur de spécifier, en fonction de ses besoins, comment les données vont être :

- lues (`SelectCommand`),
- mises à jour (`UpdateCommand`),
- créées (`InsertCommand`),
- ou supprimées (`DeleteCommand`).

ainsi que deux méthodes principales permettant de charger les données depuis la base (`Fill`) ou de répercuter les changements vers la base (`Update`).

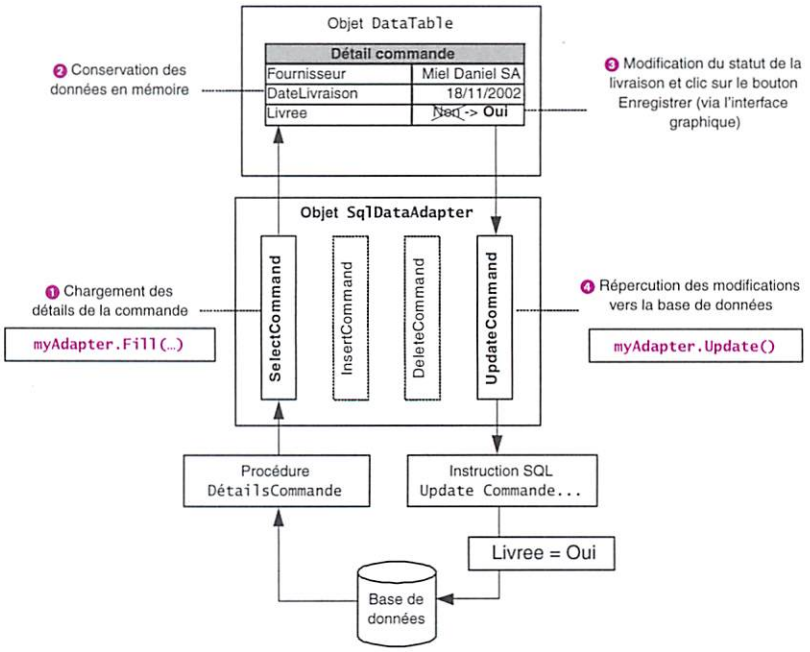


Figure 5–16 Consultation et mise à jour d'une commande avec SqlDataAdapter

Le point central du mécanisme est la classe `SqlDataAdapter` qui permet d'établir un lien entre la colonne `Livree` de l'objet `DataTable`, situé en mémoire, et la colonne correspondante de la table `Commande`, située dans la base, via le paramètre `@Livree` de la procédure stockée associée à la propriété `UpdateCommand`.

DetailCommande.aspx (Version C#)

On déclare ces deux variables au niveau de la page, afin que toutes les fonctions membres puissent y accéder.

```
SqlDataAdapter    myAdapter;
DataTable         myDataTable;

void Page_Load(Object sender, EventArgs e)
{
```

Récupération de la connexion, stockée dans l'objet `Session`.

```
   SqlConnection myConnection;
   myConnection = (SqlConnection)Session["myConnection"];
```

Récupération du numéro de commande, passé en paramètre sur la ligne de commande.

```
    ID   = Request.Params["id"];
```

Définition des commandes SQL qui vont être utilisées pour les opérations de sélection (appel de la procédure stockée `DetailsCommande`) et de mise à jour (procédure stockée in-line effectuant la mise à jour du statut de la livraison).

```
    string SelectSQL = "DetailsCommande";
    string UpdateSQL = "UPDATE Commande Set Livree = @Livree"
    UpdateSQL = UpdateSQL + " WHERE ID_Commande  = "+ID;
```

Allocation d'une instance de `SqlDataAdapter` (la commande `Select` en passée en paramètre du constructeur).

```
    myAdapter = new SqlDataAdapter(SelectSQL,myConnection);
```

```
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
    myAdapter.SelectCommand.Parameters.Add("@CommandeID",SqlDbType.Int);
    myAdapter.SelectCommand.Parameters["@CommandeID"].Value = ID;
```

On spécifie que la commande `Select` prend en paramètre le numéro de la commande.

```
    myDataTable = new DataTable();
    myAdapter.Fill(myDataTable);
```

On alloue et on remplit une instance de `DataTable` (il est indispensable d'effectuer cette opération avant la définition de la commande `Update`, qui nécessite que le schéma de la table soit créé).

```
    myAdapter.UpdateCommand = new SqlCommand(UpdateSQL,myConnection);
    myAdapter.UpdateCommand.Parameters.Add("@Livree",SqlDbType.Bit,1,
    ➥ "Livree");
```

On définit la commande `Update`, en liant le paramètre `@Livree` à la valeur de la colonne `Livree` de l'objet `DataTable` (c'est la clé du mécanisme !).

```
    if (!IsPostBack)
    {
```

Lors du premier chargement, on affiche les détails liés à la commande.

```
        DataRow CurrentRow = myDataTable.Rows[0];
```

On déclare une variable intermédiaire à des fins de simplification.

```
        NoCommande.Text = ID;

        NomFournisseur.Text = (string)CurrentRow["NomFournisseur"];
        DateLivraison.Text = String.Format("{0:d}",
        ➥ CurrentRow["DateLivraison"]);

        bool Livree = (bool)CurrentRow["Livree"];
        StatutLivraison.SelectedIndex = ( (Livree == true)? 0 : 1);
```

On récupère les informations de l'en-tête de la commande.

```
        string SQL = "ListeLignesCommandes";
        SqlCommand myCommand = new SqlCommand(SQL,myConnection);
        myCommand.CommandType = CommandType.StoredProcedure;
        myCommand.Parameters.Add("@CommandeID",SqlDbType.Int).Value = ID;
        SqlDataReader myReader = myCommand.ExecuteReader();
        LignesCommande.DataSource = myReader;
        LignesCommande.DataBind();
        myReader.Close();
    }
}
```

On récupère les lignes de commandes (rien de nouveau dans la syntaxe d'utilisation du contrôle `DataGrid`).

```
void OnEnregistrerClicked(Object sender, EventArgs e)
{
```

Ce gestionnaire d'événement est appelé lorsque l'utilisateur clique sur le bouton Enregistrer.

```
    if(StatutLivraison.SelectedIndex == 0)
    {
        myDataTable.Rows[0]["Livree"]=true;
    }
    else
    {
        myDataTable["Commande"].Rows[0]["Livree"]=false;
    }
```

On met à jour la valeur de la colonne `Livree` en fonction de l'option sélectionnée dans le contrôle `StatutLivraison`.

```
    myAdapter.Update(myDataTable);
    Response.Redirect("fournisseurs.aspx");
}
```

On enregistre les modifications apportées à `DataTable` dans la base de données puis on redirige l'utilisateur vers la page d'accueil du module.

Ce gestionnaire d'événement est appelé lorsque l'utilisateur clique sur le bouton Annuler.

On redirige l'utilisateur vers la page d'accueil du module, sans effectuer de modification.

```
void OnAnnulerClicked(Object sender, EventArgs e)
{

   Response.Redirect("fournisseurs.aspx");
}
```

La version VB.NET de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/vb/DetailCommande.aspx

Le résultat de la consultation du détail d'une commande avec modification du statut de la livraison est illustré à la figure 5-17.

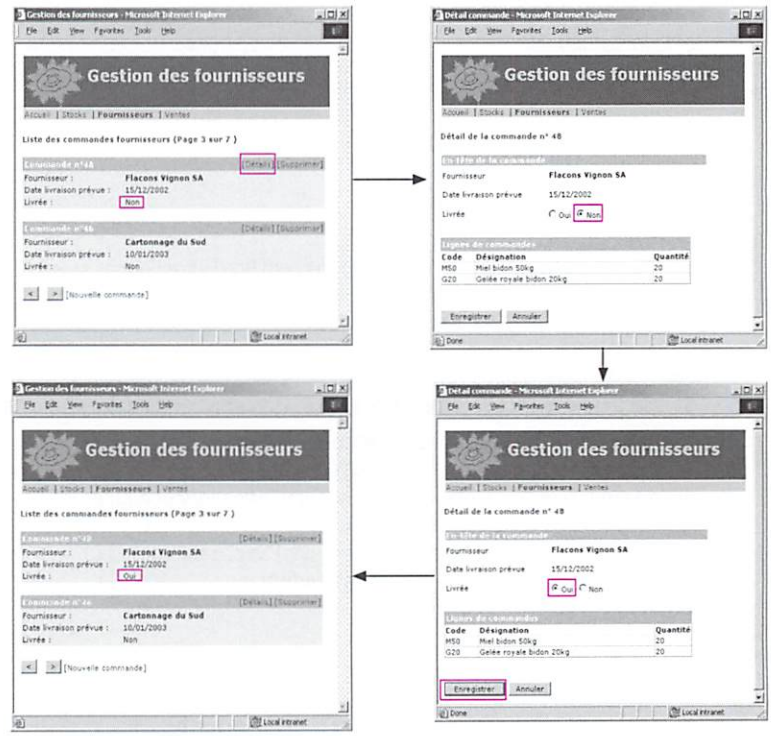


Figure 5–17 Processus de consultation et modification d'une commande

Notre module de gestion des fournisseurs est pratiquement terminé : il reste encore à implémenter l'ajout de nouvelles commandes.

Ajout d'une nouvelle commande

L'utilisateur doit pouvoir créer une nouvelle commande en cliquant sur le lien situé en bas de la page d'accueil du module : le formulaire de saisie doit permettre de choisir le fournisseur, de spécifier la date de livraison souhaitée et le détail des références commandées.

Nous allons implémenter ce mécanisme dans la page `NouvelleCommande.aspx` en utilisant une nouvelle fois la classe `SqlDataAdapter` pour réaliser la mise à jour de la base ainsi que des contrôles serveur de validation pour vérifier les informations saisies par l'utilisateur.

Réalisation de la maquette de l'interface

Outre l'utilisation d'une liste déroulante (contrôle de type `DropDownList`) pour afficher la liste des fournisseurs, cette page va comporter un certain nombre d'éléments nouveaux (voir figure 5-18) :

- un contrôle serveur de type `Calendar` (calendrier) permettant de saisir la date de livraison souhaitée ;
- un contrôle serveur de type `DataGrid` comprenant une colonne de type `TemplateColumn`, à l'intérieur de laquelle il est posssible de placer un contenu HTML spécifique (en l'occurrence, une zone d'édition pour la saisie des quantités commandées).

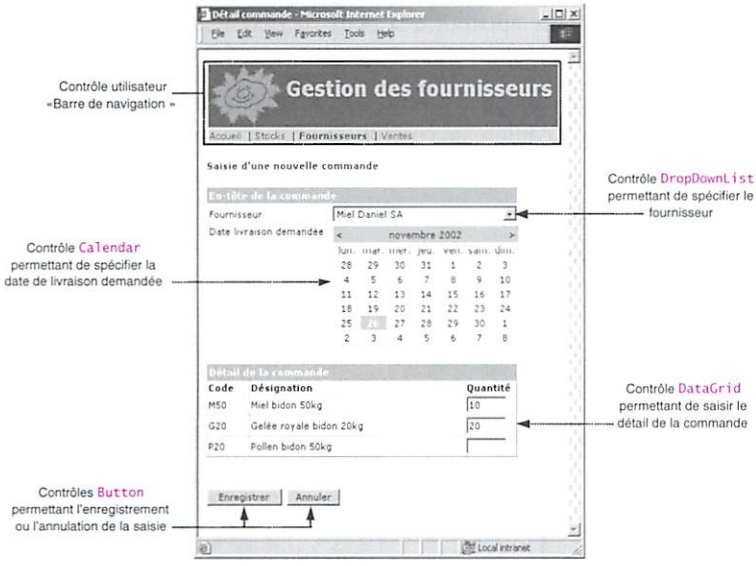


Figure 5–18 La page de saisie d'une nouvelle commande

Voici la marche à suivre pour la réalisation de la maquette :

1. Démarrez Web Matrix et ouvrez le fichier `NouvelleCommande.aspx` créé précédemment.
2. Saisissez le titre « Saisie d'une nouvelle commande ».
3. Créez un tableau HTML composé de trois lignes : une ligne de titre (« En-tête de la commande ») comportant une seule colonne et deux lignes comportant deux colonnes permettant respectivement la sélection du fournisseur

Le contrôle serveur Calendar

Ce contrôle permet la visualisation et la saisie des dates grâce à une interface présentant un calendrier mensuel ; disposant de nombreuses options de formatage, ce contrôle serveur n'en reste pas moins traduit en HTML standard dans la page Web générée !

(contrôle serveur `ListeFournisseurs` de type `DropDownList`) et la saisie de la date de livraison souhaitée (contrôle serveur `Calendrier` de type `Calendar`).

4. Créez un second tableau HTML composé d'une seule ligne et d'une seule colonne, contenant le texte « Détail de la commande ».
5. Immédiatement, en dessous, insérez un contrôle serveur de type `DataGrid` nommé `DetailCommande`.
6. À l'aide de l'assistant Property Builder de Web Matrix, désactivez l'option Create columns automatically at run time pour ce contrôle et ajoutez trois colonnes ayant les caractéristiques suivantes :

Type de colonne	Header Text	Data Field
`BoundColumn`	Code	Code
`BoundColumn`	Désignation	Désignation
`TemplateColumn`	Quantité	N/A

7. Sélectionnez le contrôle `DetailCommande` puis choisissez Edit Template dans le menu Edit afin de faire apparaître la boîte de dialogue permettant de spécifier le contenu de la colonne Quantité (voir figure 5-19).
8. Dans cette boîte de dialogue, sélectionnez l'élément ItemTemplate puis faites glisser, dans la zone d'édition, un contrôle serveur de type `TextBox` nommé `Qte`.
9. Enfin, ajoutez deux contrôles serveur de type `Button` respectivement nommés `Enregistrer` et `Annuler`.

La maquette résultante est représentée à la figure 5-20.

Les colonnes de type Template

La fonctionnalité `TemplateColumn` du contrôle `DataGrid` permet de définir une colonne dont le contenu HTML est défini par l'utilisateur (Web Matrix propose une interface graphique pour faciliter l'édition de colonnes de ce type).

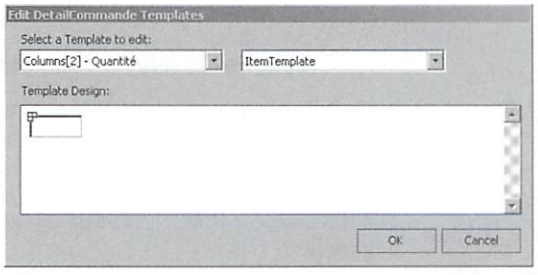


Figure 5–19 La boîte de dialogue Edit Template

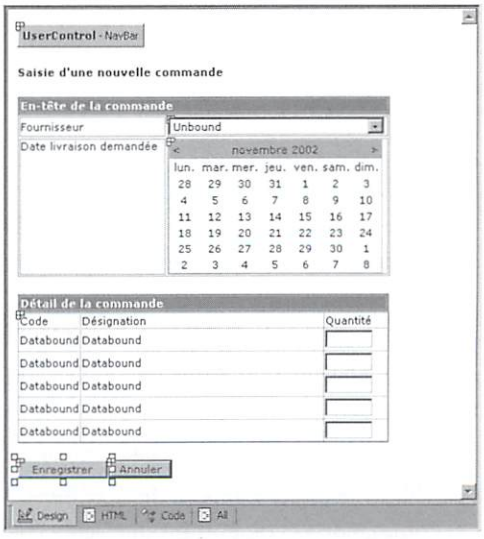


Figure 5–20 Maquette de la page de saisie d'une nouvelle commande

Afin de pouvoir implémenter le rechargement de la liste des références lorsque la sélection du fournisseur change, l'enregistrement de la commande dans la base lorsque l'utilisateur clique sur Enregistrer et le retour à la liste des commandes lorsque l'utilisateur clique sur Annuler, déclarez trois gestionnaires d'événements :

Contrôle	Événement	Gestionnaire
ListeFournisseurs	SelectedIndexChanged	OnFournisseurChanged
Enregistrer	Click	OnEnregistrerClicked
Annuler	Click	OnAnnulerClicked

Assurez-vous que la propriété AutoPostBack vaut True pour le contrôle ListeFournisseurs, afin de provoquer le rechargement automatique de la page en cas de changement de sélection du fournisseur.

Voici le code HTML correspondant (les contrôles serveur sont indiqués en couleur) :

NouvelleCommande.aspx (partie graphique)

```
...
<form runat="server">
  <h4>Saisie d'une nouvelle commande</h4>
  <table bordercolor="black">
    <tr bgcolor="#aaaadd">
      <td width="381" colspan="2">
        <font color="white"><b>En-tête de la commande</b></font>
      </td>
    </tr>
    <tr>
      <td width="150">Fournisseur</td>
      <td width="222">
        <asp:DropDownList id="ListeFournisseurs" runat="server"
          OnSelectedIndexChanged="OnFournisseurChanged"
          AutoPostBack="True"/>
      </td>
    </tr>
    <tr>
      <td valign="top" width="150">Date livraison demandée</td>
      <td width="222"><asp:Calendar id="Calendrier"/></td>
    </tr>
  </table>
  <br/>
  <table bordercolor="black" bgcolor="#aaaadd">
    <tr>
      <td width="380">
        <font color="white"><b>Détail de la commande</b></font>
      </td>
    </tr>
  </table>
```

```
  <asp:DataGrid id="DetailCommande" runat="server"
       AutoGenerateColumns="False">
    <Columns>
       <asp:BoundColumn DataField="Code" HeaderText="Code" />
       <asp:BoundColumn DataField="Designation"
            HeaderText="Designation" />
       <asp:TemplateColumn HeaderText="Quantite">
         <ItemTemplate>
            <asp:TextBox id="Qte" Width="50px" runat="server"/
         </ItemTemplate>
       </asp:TemplateColumn>
    </Columns>
   </asp:DataGrid>
   <br />
  <asp:Button id="Enregistrer" onclick="OnEnregistrerClicked"
       runat="server" Text="Enregistrer"/>
  <asp:Button id="BoutonAnnuler" onclick="OnAnnulerClicked"
       runat="server" Text="Annuler"/>
</form>
...
```

La partie graphique de la page étant terminée, passons à l'implémentation du code.

Implémentation de l'ajout d'une nouvelle commande avec SqlDataAdapter

La problématique de l'insertion d'une nouvelle commande dans la base de données va nous permettre d'illustrer pleinement le caractère « déconnecté » de la bibliothèque ADO.NET et l'utilisation des possibilités des classes `DataSet` et `SqlDataAdapter` : le principe général étant d'utiliser l'objet `DataSet` pour créer une petite base de données en mémoire, dans laquelle on insérera les nouvelles données, avant de le synchroniser avec la véritable base de données (la connexion à la base n'étant théoriquement pas requise entre ces deux étapes).

La création d'une commande concerne deux tables :

- la table `Commande` dans laquelle sera inséré l'en-tête de la commande ;
- la table `LigneCommande` dans laquelle seront insérées les lignes de commandes associées.

Nous utiliserons par conséquent deux instances de la classe `SqlDataAdapter` (une pour chaque table) qui nous permettront, lors du chargement de la page, d'initialiser le schéma du `DataSet` à partir de la base, puis, lorsque l'utilisateur cliquera sur Enregistrer, de recopier dans la base les données présentes en mémoire (voir figure 5-21).

Le point central du mécanisme est la classe `SqlDataAdapter` qui permet d'établir un lien entre les colonnes des tables situées en mémoire dans le `DataSet` et les colonnes correspondantes de la base de données grâce aux paramètres de la procédure associée à la propriété `InsertCommand`.

RAPPEL DataSet et SqlDataAdapter

Véritable mini-base de données en mémoire, un `DataSet` peut contenir plusieurs tables ; il peut être utilisé en mode déconnecté, son chargement initial et la transmission des modifications effectuées vers la base étant assurés par des objets `SqlDataAdapter` (un par table). Notons au passage qu'il n'est pas obligatoire de définir toutes les commandes du `SqlDataAdapter` mais uniquement celles qui seront utilisées (en l'occurrence `Select` et `Insert`).

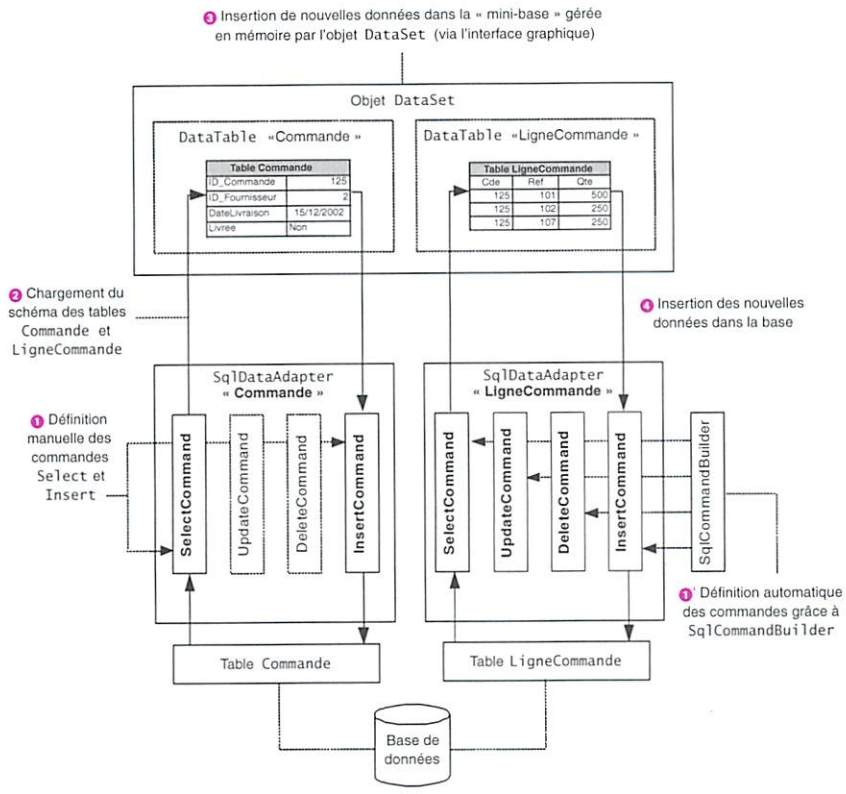


Figure 5–21 Création d'une nouvelle commande avec SqlDataAdapter

Nous associerons la propriété `InsertCommand` de l'adaptateur associé à la table `Commande` à la procédure `CreerCommande`, qui prend en paramètre l'identifiant du fournisseur et la date de livraison souhaitée, et retourne le numéro de la nouvelle commande créée.

En ce qui concerne la table `LigneCommande` : comme nous n'avons pas besoin d'obtenir de paramètre de sortie, nous pouvons nous contenter d'utiliser une procédure stockée *in-line* réalisant un `INSERT INTO` (les paramètres `@p1`, `@p2` et `@p3` doivent être, par ailleurs, respectivement associés aux colonnes `ID_Commande`, `ID_ReferenceFournisseur` et `Quantite` de la table `LigneCommande` du `DataSet`) :

```
INSERT INTO LigneCommande
   (ID_Commande, ID_ReferenceFournisseur, Quantite)
   VALUES (@p1, @p2, @p3 )
```

Plutôt que de déclarer manuellement cette procédure, nous allons utiliser la classe utilitaire `SqlCommandBuilder`, qui permet de générer automatiquement les commandes `Insert`, `Update` et `Delete` associées à une table, à partir de la commande `Select`.

La classe SqlCommandBuilder

Cette classe utilitaire génère automatiquement les commandes `Insert`, `Update` et `Delete` pour un objet de type `SqlDataAdapter`, à partir de la commande `Select`. L'intérêt de cette classe est qu'elle fait gagner du temps, son inconvénient est qu'elle ne permet pas la définition de commandes spécifiques. Notons que, curieusement, la définition des commandes par `SqlCommandBuilder` n'initialise pas les propriétés `InsertCommand`, `UpdateCommand` et `DeleteCommand` du `SqlDataAdapter` : pour accéder aux commandes, il faut utiliser les méthodes `GetInsertCommand`, `GetUpdateCommand` et `GetDeleteCommand` du `SqlCommandBuilder`.

Voici le code correspondant :

NouvelleCommande.aspx (Version C#)

On déclare la connexion au niveau de la page afin que toutes les fonctions membres puissent y accéder.

```
SqlConnection     myConnection;

void Page_Load(Object sender, EventArgs e)
{
```

Récupération de la connexion stockée dans l'objet `Session`.

```
   myConnection = (SqlConnection)Session["myConnection"];
```

Lors du premier chargement de la page, on initialise la liste des fournisseurs, on charge la liste des références correspondantes et on spécifie une date de livraison par défaut (date du jour + 1 semaine).

```
   if(!IsPostBack)
   {
         ChargerListeFournisseurs();
         AfficherListeReferences();
         Calendrier.SelectedDate = DateTime.Now.AddDays(7);
   }
}
```

Si l'utilisateur sélectionne un nouveau fournisseur, on recharge en conséquence la liste des références correspondantes.

```
void OnFournisseurChanged(Object sender, EventArgs e)
{
    AfficherListeReferences();
}
```

Cette fonction remplit le contrôle `ListeFournisseurs` à partir du contenu de la table `Fournisseur` de la base de données.

```
void ChargerListeFournisseurs()
{
    string SQL = "SELECT * FROM Fournisseur";
    SqlCommand myCommand = new SqlCommand(SQL,myConnection);
    SqlDataReader myDataReader = myCommand.ExecuteReader();
    ListeFournisseurs.DataSource = myDataReader;
    ListeFournisseurs.DataTextField  = "NomFournisseur";
    ListeFournisseurs.DataValueField = "ID_Fournisseur";
    ListeFournisseurs.DataBind();
    myDataReader.Close();
}
```

Cette fonction remplit le contrôle `DetailCommande` (de type `DataGrid`) avec les éléments de la table `Reference Fournisseur` correspondant au fournisseur sélectionné.

```
void AfficherListeReferences()
{
    string ID  = ListeFournisseurs.SelectedItem.Value;
    string SQL = "SELECT * FROM ReferenceFournisseur"
    SQL = SQL + " WHERE ID_Fournisseur = " + ID;
    SqlCommand myCommand = new SqlCommand(SQL,myConnection);
    SqlDataReader  myReader  = myCommand.ExecuteReader();
    DetailCommande.DataSource = myReader;
```

La propriété `DataKeyField` permet de spécifier le champ de la source de données à utiliser pour peupler la collection `DataKeys` associée au contrôle, qui sera utilisée lors de l'enregistrement de la commande.

```
    DetailCommande.DataKeyField = "ID_ReferenceFournisseur";
    DetailCommande.DataBind();
    myReader.Close();
}
```

Exécuté lorsque l'utilisateur clique sur le bouton Enregistrer, ce gestionnaire crée une nouvelle commande dans la base de données.

```
void OnEnregistrerClicked(Object sender, EventArgs e)
{
```

Allocation d'une instance de DataSet.

```
    DataSet myDataSet = new DataSet();
```

```
SqlDataAdapter  CommandAdapter;
string SQL;

SQL = "SELECT * FROM Commande WHERE ID_Commande = 0";
CommandAdapter = new SqlDataAdapter(SQL,myConnection);
```

Allocation du `SqlDataAdapter` utilisé pour la synchronisation des commandes (on utilise intentionnellement une instruction SQL ne correspondant à aucun enregistrement, le but étant uniquement d'initialiser le schéma du `DataSet`).

```
CommandAdapter.Fill(myDataSet,"Commande");
```

Remplissage de la table `Commande` du `DataSet` (il est nécessaire d'effectuer le remplissage avant la création de la commande `Insert`, qui requiert que le schéma soit initialisé).

```
SqlCommand myCommand;
myCommand = new SqlCommand("CreerCommande",myConnection);
myCommand.CommandType = CommandType.StoredProcedure;
SqlParameterCollections p = myCommand.Parameters;
p.Add("@FournisseurID",SqlDbType.Int,4,"ID_Fournisseur");
p.Add("@DateLivraison",SqlDbType.DateTime,8,"DateLivraison");
SqlParameter OutputParam;
OutputParam = new SqlParameter("@CommandeID",SqlDbType.Int,4);
OutputParam.Direction = ParameterDirection.Output;
p.Add(OutputParam);
CommandAdapter.InsertCommand = myCommand;
```

Définition de la commande `Insert` qui réalise la création d'une nouvelle commande : associée à la procédure `CreerCommande`, elle prend en paramètre d'entrée l'identifiant du fournisseur et la date de livraison souhaitée (respectivement liés aux colonnes `ID_Fournisseur` et `DateLivraison` du `DataSet`) et retourne en sortie l'identifiant de la commande créée.

```
SqlDataAdapter  LinesAdapter;
SQL = "SELECT * FROM LigneCommande WHERE ID_LigneCommande = 0";
LinesAdapter = new SqlDataAdapter(SQL,myConnection);
LinesAdapter.Fill(myDataSet,"LigneCommande");
```

Allocation du `SqlDataAdapter` utilisé pour la synchronisation des lignes de commandes (encore une fois, on utilise intentionnellement une instruction SQL ne correspondant à aucun enregistrement, le but étant uniquement d'initialiser le schéma du `DataSet`).

```
SqlCommandBuilder myBuilder = new SqlCommandBuilder(LinesAdapter);
```

Allocation automatique des commandes du `SqlDataAdapter` grâce à l'utilisation de la classe utilitaire `SqlCommandBuilder` (nous n'avions pas pu l'utiliser pour la table `Commande` en raison de la nécessité de recourir à une procédure stockée retournant en sortie le numéro de la commande créée).

```
DataRow myDataRow = myDataSet.Tables["Commande"].NewRow();

myDataRow["ID_Fournisseur"] = ListeFournisseurs.SelectedItem.Value;
myDataRow["DateLivraison"] = Calendrier.SelectedDate;
myDataSet.Tables["Commande"].Rows.Add(myDataRow);
```

Création d'une nouvelle ligne dans la table `Commande` du `DataSet`, initialisée à partir des valeurs des contrôles graphiques de l'interface.

```
CommandAdapter.Update(myDataSet,"Commande");
int NewCommandID = (int)OutputParam.Value;
```

Synchronisation avec la base de données (une nouvelle ligne est créée dans la table `Commande`) et récupération, grâce à `OutputParam`, de l'identifiant de la nouvelle commande.

```
for (int i = 0; i < DetailCommande.Items.Count; i++)
{
    object myControl = DetailCommande.Items[i].Cells[2].
        ➥ FindControl("Qte");
    string Qte = ((TextBox)myControl).Text;

    if (Qte!="")
    {
        myDataRow = myDataSet.Tables["LigneCommande"].NewRow();
        myDataRow["ID_Commande"] = NewCommandID;
        myDataRow["ID_ReferenceFournisseur"] =
            ➥ DetailCommande.DataKeys[i];
```

Création des lignes de commandes dans la table `LigneCommande` du `DataSet`, à partir du contenu de contrôle serveur `DetailCommande` (pour chaque ligne de la grille, on retrouve le contrôle `Qte` grâce à la fonction `FindControl` ; s'il n'est pas vide, on crée une nouvelle ligne associée à la référence fournisseur correspondante, dont l'identifiant est retrouvé grâce à `DataKeys`).

```
            myDataRow["Quantite"] =  System.Convert.ToInt32(Qte);
            myDataSet.Tables["LigneCommande"].Rows.Add(myDataRow);
        }
    }

    LinesAdapter.Update(myDataSet,"LigneCommande");


    Response.Redirect("Fournisseurs.aspx");
}

void OnAnnulerClicked(Object sender, EventArgs e)
{
    Response.Redirect("fournisseurs.aspx");
}
```

Synchronisation avec la base de données (les nouvelles lignes de commande sont créées dans la table `LigneCommande`).

Redirection vers la page d'accueil.

Exécuté lorsque l'utilisateur clique sur le bouton Annuler, ce gestionnaire retourne à la page d'accueil du module sans enregistrer la nouvelle commande.

La version VB.NET de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/vb/NouvelleCommande.aspx

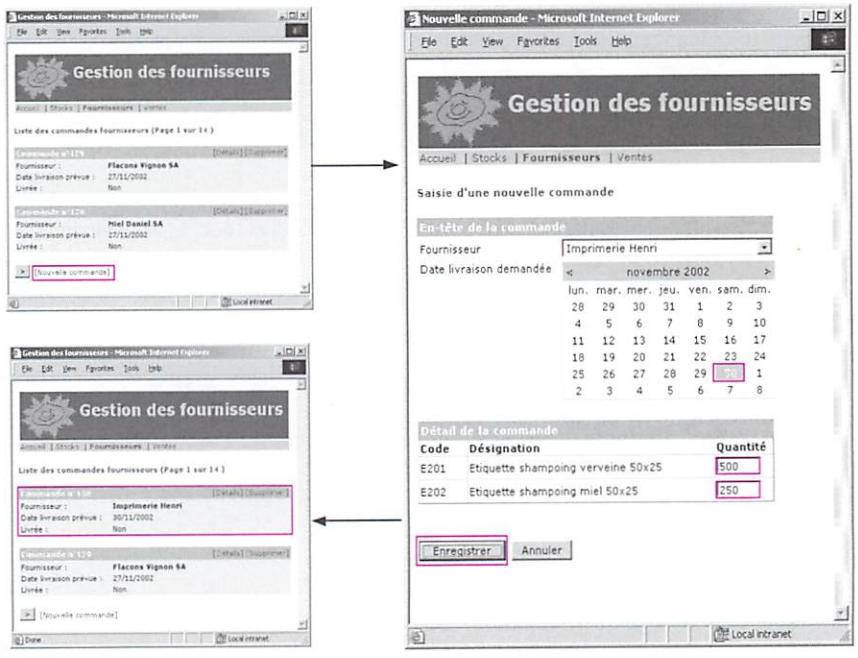


Figure 5–22 Mécanisme de création d'une nouvelle commande

Nous allons à présent améliorer cette première version (illustrée à la figure 5-22) en ajoutant un mécanisme de validation des informations saisies par l'utilisateur.

Validation des informations saisies par l'utilisateur

Une application correctement implémentée doit être dotée d'un mécanisme de validation des saisies de l'utilisateur : en l'occurrence, il faudrait vérifier que les quantités de produit saisies par l'utilisateur sont des valeurs numériques positives lors de l'envoi du formulaire HTML.

Nous allons heureusement pouvoir réaliser cette tâche facilement, grâce aux contrôles serveur de validation.

Dans notre cas, nous allons utiliser des contrôles de validation de type `RegularExpressionValidator` pour vérifier que les quantités saisies par l'utilisateur sont des entiers positifs, ainsi qu'un contrôle de type `ValidationSummary` pour récapituler la liste des erreurs :

1. Démarrez Web Matrix et ouvrez la page `NouvelleCommande.aspx`.
2. En vous plaçant dans l'onglet Design, sélectionnez le contrôle `DetailCommande` et choisissez Edit Templates dans le menu Edit.
3. Dans la boîte de dialogue qui apparaît, sélectionnez ItemTemplate et faites glisser depuis la barre d'outils Web Controls un contrôle de type `RequiredFieldValidator` (voir figure 5-23).

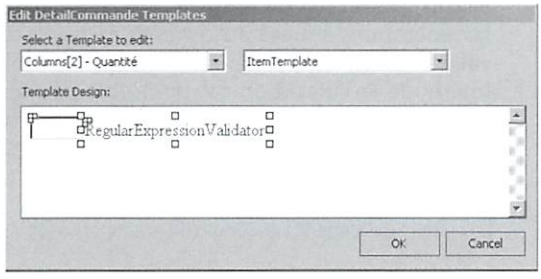


Figure 5–23 Ajout d'un contrôle de validation

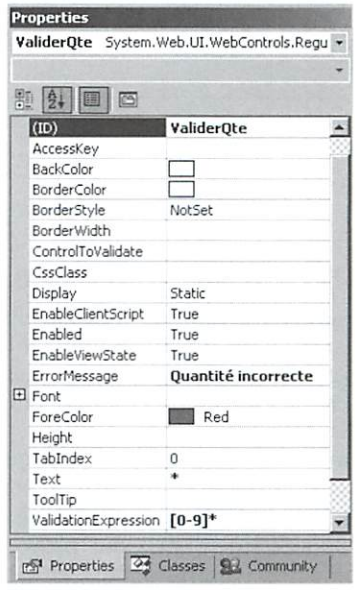


Figure 5–24 Propriétés du contrôle de validation

4. Sélectionnez le contrôle de validation et paramétrez ses caractéristiques à l'aide de la feuille de propriétés associée (voir figure 5-24).

Propriété	Valeur	Commentaire
(ID)	ValiderQte	Le nom du contrôle de validation
ControlToValidate	Qte	Le nom du contrôle à valider
Text	*	L'apparence du contrôle
ValidationExpression	[0-9]*	L'expression régulière utilisée pour la validation

5. Basculez dans l'onglet HTML pour spécifier le message d'erreur associé au contrôle, tenant compte du code fournisseur (comme au début de ce chapitre, nous faisons appel à la classe `DataBinder`).

Expressions régulières

Une expression régulière est un motif permettant de décrire un groupe de chaînes de caractères vérifiant une forme donnée : par exemple, « [0-9]* » désigne l'ensemble des chaînes de caractères composées uniquement de chiffres.

La propriété CausesValidation

Si le contenu d'un formulaire n'est pas valide, le script client de validation rend inactif les boutons qui réalisent l'envoi du formulaire, sauf ceux dont la propriété `CausesValidation` est positionnée à `False`.

ALLER PLUS LOIN Validation plus complète

Lors de la phase de validation, on pourrait également vérifier que l'utilisateur a renseigné au moins une ligne de commande. La version de l'étude de cas disponible en ligne exécute cette vérification supplémentaire.

```
<asp:RegularExpressionValidator ...
   ErrorMessage=<%# DataBinder.Eval(Container.DataItem,
      ➥ "Code","{0} : quantité incorrecte")%>
>
```

6. Ajoutez un contrôle de type `ValidationSummary` sous la grille de saisie de données et spécifiez « Merci de corriger les erreurs suivantes » pour la propriété `HeaderText` (voir figure 5-25).
7. Enfin, positionnez à `False` la propriété `CausesValidation` du bouton Annuler afin de permettre l'exécution du gestionnaire `OnAnnulerClicked`, même si le contenu du formulaire n'est pas valide.

Et voilà ! Grâce aux contrôles de validation, nous avons doté notre page de saisie de commande d'un mécanisme de vérification des saisies de l'utilisateur sans développer une seule ligne de code (voir figure 5-26).

Aller plus loin : gestion de transactions

Dans notre code de création d'une nouvelle commande dans la base, nous procédons en deux étapes : création de l'en-tête de la commande, puis création des lignes de commande. Si une erreur se produit après la création de l'en-tête mais avant la fin de la création des lignes, la commande risque d'être enregistrée de manière incomplète, laissant la base dans un état incohérent.

On ne peut pas écarter le risque d'une panne extérieure à l'application (disque dur plein, temps de réponse de la base trop important, etc.), en revanche, on peut éviter de laisser la base dans un état incohérent lorsqu'une panne survient en faisant appel à une transaction, qui peut être gérée soit au sein de la base de données, soit au niveau applicatif, grâce à l'objet `SqlTransaction` de la bibliothèque ADO.NET.

Transaction

Une transaction est un regroupement logique d'opérations qui doivent être effectuées de manière atomique : soit une transaction commencée (`begin`) s'achève correctement (`commit`), soit toutes les opérations entreprises sont annulées (`rollback`). Outre l'atomicité, une transaction assure également l'isolation des données (deux transactions ne peuvent pas accéder simultanément aux mêmes données), leur cohérence (une transaction, réussie ou échouée, laissera toujours la base dans un état cohérent) et la persistance des opérations effectuées (en cas de panne, les actions en attente sont traitées au redémarrage).

Le principe général est de commencer la transaction, en obtenant un objet `SqlTransaction` via la fonction `BeginTransaction` de la classe `SqlConnection`, d'effectuer les opérations situées au cœur de la transaction, puis de valider la transaction (méthode `Commit`) ou de l'annuler (méthode `Rollback`) en cas de problème ; tous les objets accédant aux données doivent être liés à la transaction via la propriété `Transaction` de la classe `SqlCommand` :

```
SqlTransaction myTransaction =
   ➥ myConnection.BeginTransaction();
try
{
  ...
  myCommand = new SqlCommand(SQL,myConnection);
  myCommand.Transaction = myTransaction;
  ...
}
catch(Exception myException)
{
   myTransaction.Rollback();
}

myTransaction.Commit();
```

Code source complet téléchargeable en ligne

Dans cette section, nous ne faisons que donner des indications générales pour la mise en œuvre de transactions ; le code source complet (VB.NET et C#) est disponible sur le site www.savonsdusoleil.com.

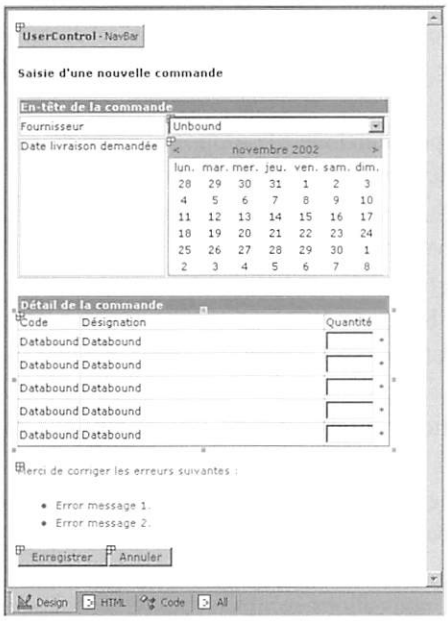


Figure 5–25 Ajout d'un contrôle de type `ValidationSummary`

Figure 5–26 Détection d'une erreur de saisie

Les contrôles serveur de validation

Les contrôles serveur de validation permettent de valider simplement les informations saisies par l'utilisateur dans un formulaire Web : chaque contrôle de validation est associé à un contrôle de type « champ de saisie » (`TextBox`, `DropDownList`...) dont il vérifie la valeur ; en cas de saisie incorrecte, les contrôles de validation – masqués par défaut – s'affichent pour signaler les erreurs (chaque contrôle est généralement placé à côté du champ de saisie qu'il valide).

Il existe plusieurs types de contrôles de validation, qui diffèrent suivant le type de vérification effectué sur la valeur du contrôle auquel ils sont associés :

- `RequiredFieldValidator` vérifie qu'une valeur est présente ;
- `RangeValidator` vérifie que la valeur est contenue dans une certaine fourchette ;
- `CompareValidator` compare la valeur à celle d'un autre contrôle, ou à une valeur absolue ;
- `RegularExpressionValidator` teste si la valeur vérifie une expression régulière donnée ;
- `CustomValidator` teste si la valeur vérifie une règle spécifique fixée par le développeur.

À cette liste, il faut ajouter le contrôle `ValidationSummary` qui permet de faire la synthèse des erreurs de validation présentes sur une page.

Deux propriétés contrôlent l'affichage des messages d'erreur :

- la propriété `Text` permet de spécifier l'apparence du contrôle de validation (par exemple, une étoile rouge), qui s'affichera à côté du champ fautif ;
- la propriété `ErrorMessage` permet de spécifier un message d'erreur associé au contrôle (par exemple : « Entrez une adresse e-mail valide »), qui sera affiché, le cas échéant, dans le contrôle `ValidationSummary` présent sur la page).

Par défaut, la validation est effectuée au niveau du client (à moins que le navigateur ne soit pas compatible avec les scripts ou que la propriété `EnableClientScript` d'un des contrôles de validation ait été positionnée à `False`), sinon, elle est effectuée au niveau serveur.

Plus d'informations sur le mécanisme de validation

Notons que dans le cas d'une validation intervenant côté client, le moteur ASP.NET adapte le script en fonction du navigateur ; dans tous les cas, la validation a lieu une nouvelle fois côté serveur afin d'éviter les risques de piratage.

En résumé...

Dans ce chapitre, nous avons présenté un grand nombre de techniques, que nous avons mises en œuvre pour réaliser le module de gestion des fournisseurs de notre étude de cas :

- affichage d'une liste personnalisée de données, suivant une maquette HTML définie par le développeur, à l'aide du contrôle `Repeater` ;
- gestion d'événements émis par des contrôles situés au sein d'un contrôle principal (remontée d'événements « en bulle ») ;
- utilisation du couple `SqlDataAdapter/DataSet` pour mettre à jour ou ajouter des données dans une base, en définissant manuellement les commandes SQL associées ou ayant recours à la classe `SqlCommandBuilder` pour les définir automatiquement ;
- validation des informations saisies par l'utilisateur grâce aux contrôles serveur de validation.

Au passage, nous avons également indiqué comment implémenter une liste paginée avec `PagedDataSource` et mettre en œuvre des transactions avec `SqlTransaction`.

Dans le prochain chapitre, nous allons compléter notre module de gestion des commandes fournisseur en réalisant la génération et l'envoi d'un fichier XML pour chaque nouvelle commande.

Échanges XML avec les fournisseurs

6

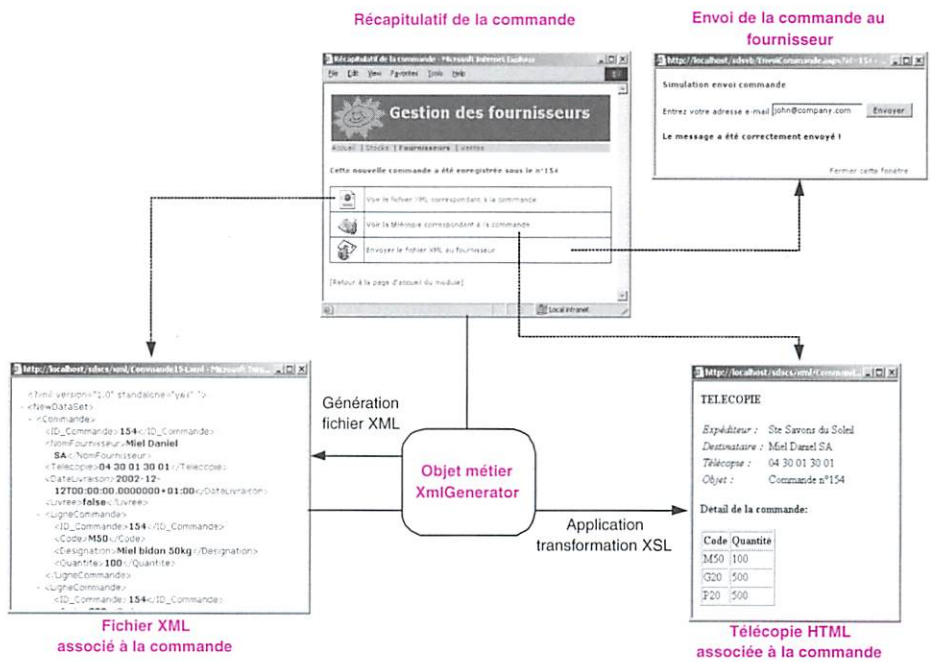
ASP.NET

Compilateurs vbc et csc | WriteXml | XmlDocument | XslTransform | MailMessage | SmtpMail

SOMMAIRE

MOTS-CLÉS

- Compilateurs vbc et csc
- WriteXml
- XmlDocument
- XslTransform
- MailMessage
- SmtpMail



Dans ce chapitre, nous présentons quelques-unes des nombreuses fonctionnalités relatives à XML disponibles au sein d'ASP.NET (génération de documents XML, application de transformations XSL) que nous mettons en œuvre à travers un objet métier, afin de perfectionner le module de gestion des fournisseurs développé au chapitre précédent, en lui ajoutant la génération et l'envoi d'un fichier XML pour chaque nouvelle commande saisie.

Préparation de l'ajout des fonctionnalités XML

Si les fonctionnalités développées au chapitre 5 (consultation, ajout, mise à jour et suppression de commandes fournisseur) apportent un bénéfice en terme d'organisation interne, elles ne simplifient pas les échanges avec les fournisseurs : nous allons maintenant combler cette lacune en implémentant l'envoi automatique de chaque nouvelle commande au fournisseur concerné sous la forme d'un fichier XML.

Comme certains fournisseurs souhaitent continuer à recevoir les commandes par télécopie, nous implémenterons également la génération automatique d'une télécopie correspondant à la commande, en utilisant une transformation XSL appliquée au fichier XML.

Ces fonctionnalités étant indépendantes de la couche de présentation graphique, nous les encapsulerons au sein d'un composant indépendant, ce qui les rendra, au passage, plus facilement réutilisables.

Dans la version précédente de notre module, un utilisateur venant de saisir une commande était redirigé vers la liste des commandes : nous allons désormais le rediriger vers une page récapitulative permettant de consulter les fichiers générés suite à la saisie, dont nous commencerons, comme d'habitude, par réaliser la maquette.

Réalisation de la maquette de la page récapitulative de la commande

On souhaite qu'à la fin du processus de saisie d'une commande, l'utilisateur soit redirigé vers une page récapitulative permettant :

- la consultation du fichier XML correspondant à la commande ;
- la consultation de la télécopie correspondant à la commande, générée sous forme HTML à partir du fichier XML de la commande, par le biais d'une transformation XSL ;
- la simulation de l'envoi du fichier XML au fournisseur.

Comme nous avons prévu d'implémenter la génération des fichiers dans un composant indépendant, cette page se résumera principalement à un ensemble de liens (voir figure 6-1).

Par conséquent, la réalisation de la maquette ne comporte aucune difficulté particulière :

1. Démarrez Web Matrix.
2. Créez une nouvelle page nommée `FinCommande.aspx`.
3. Placez-vous dans l'onglet HTML et spécifiez le titre ainsi que le lien vers la feuille de style de l'application, dans l'en-tête de la page (section `<head>`) :

```
<title>Récapitulatif de la commande</title>
<link href="SDS.css" type="text/css" rel="stylesheet" />
```

À propos d'XML

On ne présente plus XML, un langage permettant de décrire des structures de données et des documents de tout type, à l'aide de balises personnalisées : il est de plus en plus utilisé par toutes les applications qui manipulent, stockent ou échangent des données.

📖 *Initiation à XML*, D. Hunter, Eyrolles 2001

À propos des transformations XSL

Les transformations XSL sont un mécanisme puissant qui permet de produire des fichiers de formats divers (XML, HTML, XHTML, RTF) à partir de données contenues dans un fichier XML.

📖 *XSLT fondamental*, P. Drix, Eyrolles 2002

📖 *XSLT par la pratique*, S. Holzner, Eyrolles 2002

DANS UN CAS RÉEL

Infrastructure requise chez le fournisseur

Contrairement aux protocoles d'échanges interapplicatifs spécifiques, XML n'impose aucune contrainte technique complexe (paramétrage de pare-feu, intégration avec les applications existantes, etc.) : le fournisseur doit uniquement disposer d'un logiciel capable de traiter le fichier XML reçu et, bien entendu, avoir convenu d'un format d'échange (schéma) avec l'émetteur du fichier. Les données XML peuvent être reçues soit par messagerie (le cas que nous développons dans ce chapitre), soit directement par appel d'un service Web (voir chapitre 8).

DANS UN CAS RÉEL

Envoi automatique du fichier

Dans un cas réel, le fichier serait probablement transmis de manière automatique : dans notre étude de cas, nous réaliserons l'envoi du fichier sur demande de l'utilisateur, à une adresse de son choix, de manière à simuler l'envoi du fichier au fournisseur.

Figure 6–1 La page récapitulative de la commande

4 Au début du corps de la page (section <body>), insérez une référence au contrôle utilisateur « Barre de navigation », réalisé au chapitre 3, en sélectionnant la valeur de l'index correspondant au module Fournisseurs :

```
<SDS:NavBar id="NavBar" runat="server" SelectedIndex="2"/>
```

5 Toujours au sein de l'onglet HTML, insérez un script client permettant l'affichage d'une fenêtre popup, afin de pouvoir réaliser l'affichage des fichiers :

```
<script>
function openpopup(popup_url,params)
  {
     retcode=window.open(popup_url,'',params);
  }
</script>
```

6 Placez-vous maintenant dans l'onglet All et insérez, en haut de la page, les directives permettant de faire référence au contrôle utilisateur « Barre de navigation » ainsi qu'aux classes de la bibliothèque ADO.NET, qui nous seront utiles par la suite :

```
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

7 Enfin, placez-vous dans l'onglet Design et saisissez le titre « Récapitulatif de la commande n° » suivi d'un contrôle serveur de type Label nommé NoCommande, destiné à afficher le numéro de la commande.

8 Sous ce titre, insérez un tableau HTML comportant trois lignes et deux colonnes.

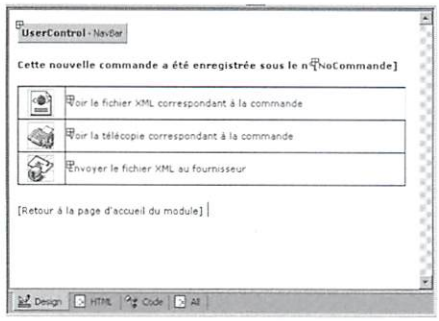


Figure 6–2 Maquette de la page récapitulative d'une commande

9 Dans la colonne de gauche, insérez trois contrôles serveur de type `Image` et faites-les pointer respectivement vers les fichiers `XML.gif`, `Fax.gif` et `Email.gif` (téléchargeables depuis le site de l'étude de cas) en utilisant la propriété `ImageUrl` des feuilles de propriétés associées aux contrôles.

10 Dans la colonne de droite, insérez trois contrôles serveur de type `HyperLink` et paramétrez-les suivant les indications fournies dans le tableau :

Ligne	Nom du contrôle	Texte du contrôle
1	VoirXML	Voir le fichier XML correspondant à la commande
2	VoirFax	Voir la télécopie correspondant à la commande
3	EnvoyerXML	Envoyer le fichier XML au fournisseur

11 Pour finir, insérez un lien hypertexte simple « Retour à la page d'accueil du module » pointant vers la page `Fournisseurs.aspx`.

La maquette correspondante est représentée figure 6-2.

Voici le contenu HTML correspondant (les contrôles serveur sont indiqués en couleur) :

FinCommande.aspx (partie graphique)

```
<html>
<head>
  <title>Récapitulatif de la commande</title>
  <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
  <SDS:NavBar id="NavBar" runat="server" SelectedIndex="2">
  </SDS:NavBar>

 <script>
  function openpopup(popup_url,params)
  {
     retcode=window.open(popup_url,'',params);
  }
 </script>

 <form runat="server">
   <h4>Cette nouvelle commande a été enregistrée sous le n°
      <asp:Label id="NoCommande" runat="server"></asp:Label>
   </h4>
   <br/>
   <table bordercolor="black" cellpadding="3" border="1">
    <tr><td width="50"><img src="img/XML.gif" /></td>
      <td width="400">
       <asp:HyperLink id="VoirXML" runat="server">
           Voir le fichier XML correspondant à la commande
       </asp:HyperLink>
       </td></tr>
    <tr><td width="50"><img src="img/XML.gif" /></td>
      <td width="400">
```

```
      <asp:HyperLink id="VoirFax" runat="server">
          Voir la télécopie correspondant à la commande
      </asp:HyperLink>
    </td></tr><tr><td width="50"><img src="img/Fax.gif" /></td>
    <td width="400">
      <asp:HyperLink id="EnvoyerXML" runat="server">
          Envoyer le fichier XML au fournisseur</asp:HyperLink>
      </td></tr><tr><td width="50"><img src="img/Email.gif" />
    </td></tr>
  </table>
    <a href="./fournisseurs.aspx">
      [Retour à la page d'accueil du module]</a>
</form>
</body>
</html>
```

Afin d'effectuer le lien entre le code existant et cette nouvelle page, modifiez le fichier `NouvelleCommande.aspx` réalisé au chapitre précédent afin qu'il redirige l'utilisateur vers la page `FinCommande.aspx`, en passant le numéro de la commande en paramètre sur la chaîne de requête :

```
Response.Redirect ("FinCommande.aspx?id="+NewCommandID)
```

La maquette de la page étant réalisée, passons maintenant à l'implémentation du code de la page, qui va réaliser la mise à jour du numéro de commande, la génération des fichiers requis et le paramétrage des liens afin qu'ils pointent vers ces fichiers.

Encapsulation des fonctionnalités XML dans un objet métier

Dans les pages ASP.NET que nous avons réalisées précédemment, l'intégralité du code était placée au sein de la page : ceci était justifié par le fait que les fonctionnalités développées relevaient principalement de la présentation de l'application.

En revanche, les fonctionnalités XML que nous nous apprêtons à mettre en place ne sont pas directement liées à l'interface graphique et relèvent plus de la logique interne de l'application : pour cette raison, il est opportun de les implémenter au sein d'un composant indépendant ou objet métier, afin de rendre notre application plus modulaire.

Nous allons donc implémenter la génération des fichiers dans un composant spécifique, que nous nommerons `XmlGenerator`, qui devra être capable de réaliser deux actions :

- génération d'un fichier XML correspondant à une commande à partir des informations contenues dans la base de données ;
- génération d'une télécopie au format HTML, par application d'une transformation XSL au fichier XML précédemment généré.

Lors de son chargement, la page récapitulative de la commande allouera une instance de cet objet métier, auquel elle sous-traitera la génération des fichiers

À propos des objets métier

Un objet métier est un composant constitutif d'une application qui encapsule des fonctionnalités liées à un processus spécifique d'une organisation. Il est conseillé d'implémenter la logique interne d'une application sous forme d'objets métier, lesquels sont plus facilement réutilisables et rendent l'application plus modulaire, donc plus facile à maintenir.

pour la commande venant d'être saisie (dont le numéro sera passé en paramètre sur la chaîne de requête), puis paramètrera ses liens hypertextes afin qu'ils pointent vers les fichiers générés (voir figure 6-3).

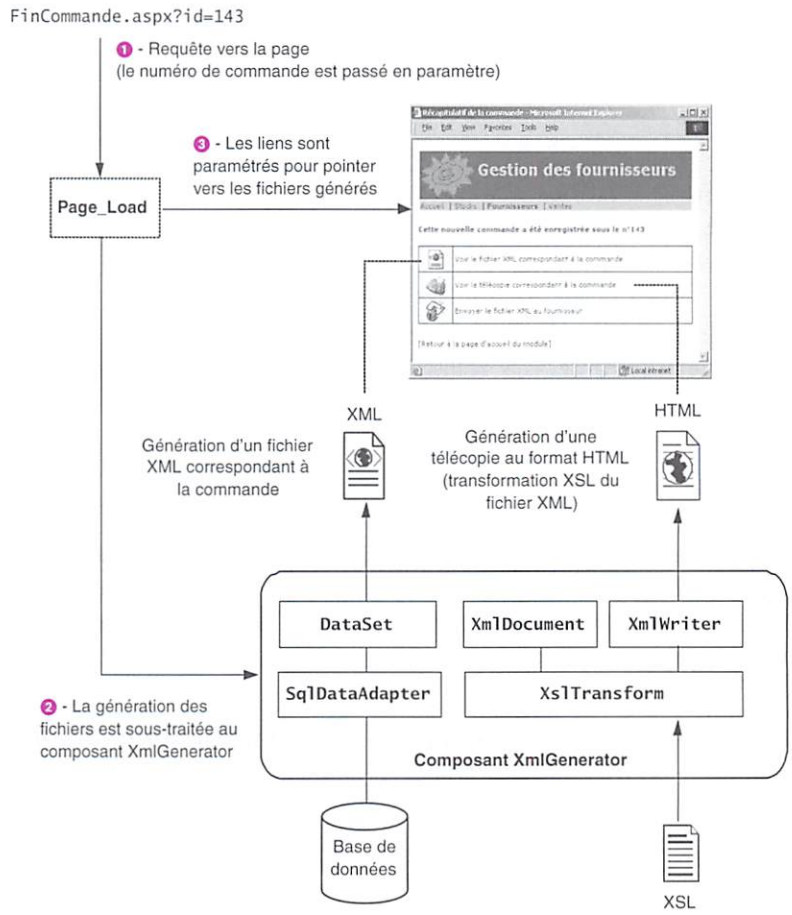


Figure 6–3 Architecture de la page récapitulative d'une commande

Nous allons, dans un premier temps, réaliser le squelette de ce composant et l'intégrer dans la page `FinCommande.aspx`, puis, nous implémenterons ses fonctionnalités proprement dites.

Création du composant XmlGenerator

D'un point de vue pratique, un objet métier est implémenté sous la forme d'une classe indépendante, dont nous allons créer simplement le squelette à l'aide de Web Matrix :

1 Choisissez New dans le menu File : une boîte de dialogue apparaît (voir figure 6-4).

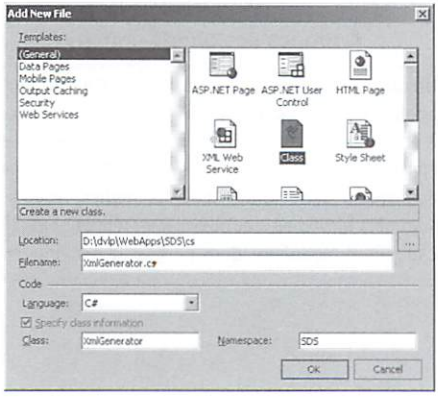


Figure 6–4 Création d'une classe avec Web Matrix

2 Sélectionnez le type de fichier Class et spécifiez :

- l'emplacement du nouveau fichier (le répertoire de votre application) ;
- son nom (par exemple : `XmlGenerator`) ;
- le langage de votre choix (C# ou VB.NET) ;
- le nom de la classe (par exemple : `XmlGenerator`) ;
- l'espace de nommage, obligatoire, dans lequel sera stockée la classe (par exemple : `SDS`, pour « Savons du Soleil »).

Sur le plan conceptuel, notre classe `XmlGenerator` devra contenir :

- un constructeur, prenant en paramètre le numéro de la commande traitée par l'instance, la connexion vers la base de données et le chemin dans lequel placer les fichiers générés (ces trois valeurs seront stockées dans des variables membres privées, afin que les autres membres de la classe puissent y faire référence) ;
- une méthode publique `GenererFichierXml` prenant en paramètre le nom du fichier à générer ;
- une méthode publique `GenererTelecopie` prenant en paramètre les noms du fichier XML source, du fichier XSL spécifiant la transformation à appliquer et du fichier HTML produit en sortie.

Voici le code correspondant, après déclaration de méthodes « vides », qui seront implémentées dans la suite de ce chapitre :

Fichier XmlGenerator.vb (Version VB.NET)

```
Option Explicit
Option Strict

Imports System
Imports System.Text
Imports System.Data
Imports System.Data.SqlClient
Imports System.Data.Xml
Imports System.Data.Xml.Xsl

Namespace SDS



  public Class XmlGenerator

    private Connection As SqlConnection
    private NoCommande As String
    private Path As String


    Public Sub New(no As String, c As SqlConnection, p As String )
        NoCommande  = no
        Connection  = c
        Path        = p
    End Sub
```

Ces deux options indiquent que toutes les variables doivent être déclarées explicitement et que les conversions de type implicite sont fortement limitées.

Contrairement à ce qui se passe pour une page ASP.NET, aucun espace de nommage n'est inclus implicitement : nous spécifions donc explicitement ceux dont nous allons avoir besoin.

Pour pouvoir faire référence à la classe ultérieurement, il faut obligatoirement inclure la classe dans un espace de nommage.

Déclaration de la classe `XmlGenerator`.

Déclaration de trois variables membres (connexion vers la base de données, numéro de la commande à traiter, répertoire d'écriture des fichiers générés).

Le constructeur de la classe attend en entrée les trois paramètres nécessaires à l'initialisation des variables membres. Il est indispensable de procéder ainsi car une classe indépendante n'a pas accès aux objets intrinsèques ASP.NET (`Session`, `Request`).

Méthode publique réalisant la génération du fichier XML correspondant à la commande (`xmlFile` désigne le nom à utiliser pour le fichier généré).

Méthode publique réalisant la génération de la télécopie HTML associée à la commande (les paramètres désignent respectivement le nom du fichier XML à transformer, le nom du fichier XSL et celui du fichier HTML de sortie).

```
    Public Sub GenererFichierXml(xmlFile As String)
        ' Sera implémentée plus tard
    End Sub


    Public Sub GenererTelecopie(xml As String, xsl As String, html As String )
        ' Sera implémentée plus tard
    End Sub

  End Class
End Namespace
```

Et voici la version C# de cette même classe :

Fichier XmlGenerator.cs (Version C#)

Le constructeur porte le nom de la classe, comme en C++ (alors qu'en VB.NET, un constructeur doit porter le nom New).

```
using System;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Xml;
using System.Xml.Xsl;

namespace SDS {

  public class XmlGenerator
  {
    private SqlConnection Connection;
    private string NoCommande;
    private string Path;
    public XmlGenerator(string no, SqlConnection c, string p)
    {
        NoCommande  = no;
        Connection  = c;
        Path        = p;
    }

    public void GenererFichierXml(string xmlFile)
    {
      // Sera implementée plus tard
    }

    public void GenererTelecopie(string xml, string xsl, string output)
    {
      // Sera implementée plus tard
    }
  }
}
```

Pour pouvoir utiliser cette classe dans notre application, nous devons préalablement la compiler : c'est ce que nous allons faire dans la section suivante.

Compilation du composant XmlGenerator

Tous les éléments d'une application ASP.NET sont compilés : en ce qui concerne les pages que nous avons développées jusqu'à présent, la compilation était effectuée à la volée ; en revanche, nous allons devoir compiler manuellement notre composant métier au sein d'un assemblage avant de pouvoir y faire référence.

La figure 6-5 illustre le processus de compilation mis en œuvre :

- lors du développement de l'application, la classe `XmlGenerator` est compilée sous la forme d'un assemblage de type bibliothèque (*library*) nommé `SDS.dll`, en raison de l'espace de nommage contenu, et placé dans le répertoire des assemblages privés de l'application (voir note) ;
- lors de la première requête vers la page `FinCommande.aspx`, qui importe l'espace de nommage `SDS`, le code de la page est compilé et lié à l'assemblage `SDS.dll` : l'assemblage temporaire résultant est placé dans le répertoire `Temporary ASP.NET Files` ;
- lors des requêtes suivantes, l'assemblage temporaire correspondant à la page, conservé en cache, est exécuté directement, sans nécessiter le recours à la compilation.

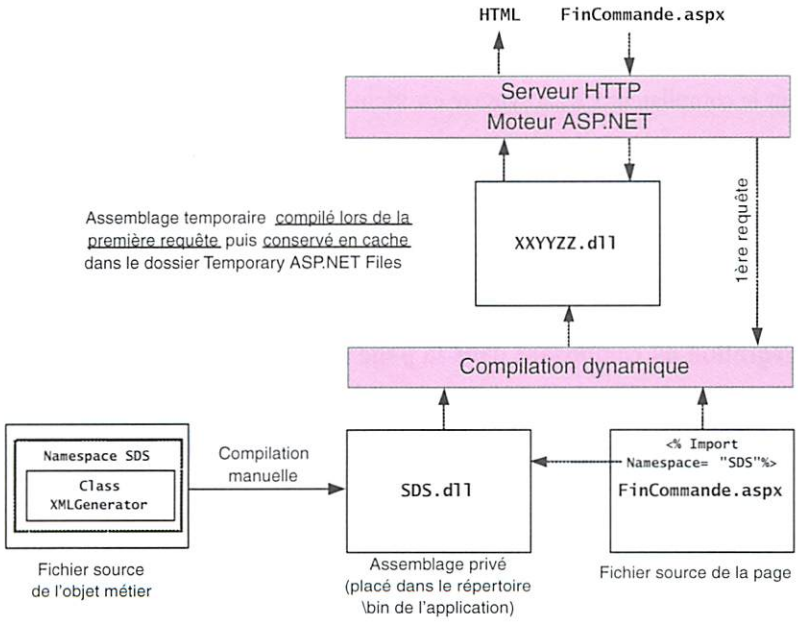


Figure 6–5 Processus de compilation d'une page utilisant un composant externe

Contrairement à Visual Studio.NET, l'environnement Web Matrix ne dispose pas de compilateur intégré : nous allons donc devoir faire appel au compilateur en mode ligne de commande, fourni avec le kit de développement .NET, au sein d'une fenêtre de commande DOS (voir figure 6-6).

RAPPEL Assemblage

Un assemblage (assembly en anglais) désigne une brique binaire élémentaire réutilisable d'une application .NET, qui correspond généralement à une hiérarchie d'espaces de nommage, contenant chacun plusieurs classes. Constitué d'un ou plusieurs fichiers de type DLL ou EXE, un assemblage a la particularité de s'autodécrire : il contient un résumé (*manifest*) qui précise notamment la liste des fichiers et des types contenus, ainsi que des informations de version et de sécurité.

Assemblages privés vs assemblages partagés

Un assemblage privé n'est visible que par une seule application (il doit être placé dans un répertoire nommé `bin` situé sous la racine de l'application) : l'avantage principal est la simplicité de déploiement (simple copie de fichier) et l'isolation (pas de risque de conflits entre applications). Un assemblage partagé peut être utilisé par plusieurs applications sur une même machine : placé dans le cache global (*Global Assembly Cache*), il doit obligatoirement être signé pour des raisons de sécurité.

Préparation de la compilation

N'oubliez pas de créer un répertoire `\bin` sous la racine de votre application. D'autre part, si vous n'avez pas choisi d'enregistrer les variables d'environnement lors de l'installation du .NET SDK, ajoutez le chemin :

`\WINNT\Microsoft.NET\Framework\v1.0.3705`

dans la variable d'environnement `Path` de votre système, afin de pouvoir faire référence au compilateur depuis n'importe quel répertoire.

La syntaxe à utiliser pour compiler une classe écrite en VB.NET est la suivante :

```
vbc /out:bin\SDS.dll /t:library XmlGenerator.vb /r:System.dll
➥ /r:System.Data.dll /r:System.Xml.dll
```

où :

- `vbc` est le nom du compilateur VB.NET ;
- `/out` indique le chemin relatif du fichier de sortie ;
- `/t` indique le type d'assemblage produit (en l'occurrence, une bibliothèque) ;
- `/r` spécifie les assemblages auxquels le fichier compilé fait référence (que nous spécifions d'ores et déjà tous, bien qu'ils ne soient pas encore tous requis, du fait des méthodes non implémentées).

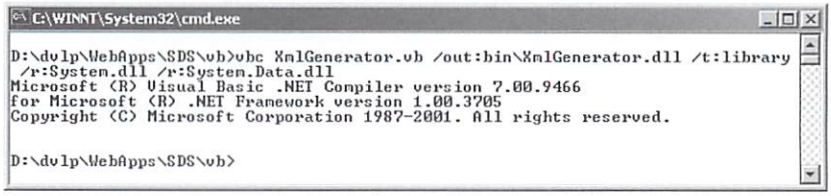


Figure 6–6 Compilation d'une classe

Pour le compilateur C#, la syntaxe est identique, au nom du compilateur près :

```
csc /out:bin\SDS.dll /t:library XmlGenerator.cs
➥ /r:System.dll /r:System.Data.dll /r:System.Xml.dll
```

Notre composant étant compilé, passons maintenant à l'implémentation de la page `FinCommande.aspx`, qui va y faire référence.

Intégration du composant dans la page récapitulative de la commande

Pour pouvoir utiliser notre composant depuis une page ASP.NET, il faut importer l'espace de nommage dans lequel il est contenu, comme nous le faisons pour les classes de la bibliothèque .NET, en plaçant la directive suivante en haut du fichier :

```
<%@ Import Namespace="SDS" %>
```

Ceci étant fait, il ne reste plus qu'à implémenter la récupération du numéro de commande passé en paramètre, l'instanciation d'un objet `XmlGenerator`, la création des fichiers et le paramétrage des liens dans le gestionnaire `Page_Load`.

Pour chaque commande, deux fichiers seront générés et placés dans un répertoire `xml` (à créer), situé sous la racine de l'application : un fichier `CommandeN.xml`, qui contient les informations de la commande au format XML, et un fichier `CommandeN.html`, qui constitue la télécopie associée à la commande (`N` désigne le numéro de la commande).

Cohabitation de plusieurs langages

Notons qu'il est tout à fait possible d'instancier un composant écrit en VB.NET depuis une page écrite en C# et vice-versa, puisque tous les objets sont compilés dans le langage intermédiaire MSIL.

Fichier FinCommande.aspx (Version VB.NET)

```
Sub Page_Load(sender As Object,e As EventArgs)

    Dim ID As String
    ID = Request.Params("id")

    NoCommande.Text = ID

    Dim path As String
    path = Request.MapPath(Request.ApplicationPath)+"\\xml\\"




    Dim myConnection As SqlConnection
    myConnection = CType(Session("myConnection"),SqlConnection)

    Dim myGenerator As XmlGenerator
    myGenerator = new XmlGenerator(ID,myConnection,path)

    Dim  xmlFile, xslFile, htmlFile As String
    xmlFile  = "Commande"+ID+".xml"
    xslFile  = "GenererTelecopie.xsl"
    htmlFile = "Commande"+ID+".html"

    myGenerator.GenererFichierXml(xmlFile)
    myGenerator.GenererTelecopie(xmlFile,xslFile,htmlFile)

    Dim PopupArgs As String

    PopupArgs = "'scrollbars=no,status=no,resizable=yes,
        ➥ width=500,height=400'"
    VoirXML.NavigateUrl="javascript:openpopup('xml/"+
        ➥ xmlFile+"','"+PopupArgs+")"
    VoirFax.NavigateUrl="javascript:openpopup('xml/"+
        ➥ htmlFile+"','"+PopupArgs+")"

End Sub
```

Récupération du numéro de commande passé en paramètre dans la chaîne de requête

Mise à jour du numéro de commande affiché

Spécification du répertoire de création des fichiers (répertoire xml situé sous la racine de l'application) ; la méthode `MapPath` de l'objet `Request` permet d'obtenir le chemin physique du répertoire de l'application

Récupération de la connexion stockée dans l'objet `Session`

Déclaration et allocation d'un objet `XmlGenerator`

Spécification des noms des fichiers concernés (le fichier `GenererTelecopie.xsl` sera créé plus loin dans ce chapitre)

Génération des fichiers (pour l'instant, les méthodes ne font rien !)

Paramétrage des liens, afin qu'ils affichent les fichiers générés dans les fenêtres de type popup

La version C# de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/cs/FinCommande.aspx

Toute la mécanique de génération des fichiers est prête, à un petit détail près : pour l'instant, notre composant ne fait rien ! Nous allons donc passer à l'implémentation du cœur de notre objet métier.

Génération des fichiers liés à la commande

Dans un premier temps, nous allons implémenter la méthode `GenererFichierXml`, qui générera un fichier XML contenant les informations de la commande, puis nous passerons à la méthode `GenererTelecopie`, qui générera une télécopie au format HTML à partir du fichier XML précédent et d'une feuille de style XSL.

DANS UN CAS RÉEL

Optimisation de l'accès aux données

Dans le module réalisé dans le chapitre précédent, les données de la commande saisie sont stockées en mémoire avant d'être écrites dans la base de données : dans un cas réel, il serait opportun de générer le fichier XML au moment de la saisie à partir des données stockées en mémoire, plutôt que de les recharger ultérieurement depuis la base, comme nous le faisons ici.

DataSet et XML

Si la génération de fichiers XML est si simple, c'est parce que l'architecture interne de stockage de la classe `DataSet` est entièrement basée sur XML. Signalons au passage l'existence de la méthode `WriteXmlSchema`, qui permet de générer automatiquement un schéma XML (fichier `.xsd`) correspondant à la structure interne d'un `DataSet`.

Génération du fichier XML à partir d'un objet DataSet

La génération d'un fichier XML contenant les informations de la commande est extrêmement simple à implémenter, puisqu'elle peut être réalisée en une seule ligne de code grâce à la méthode `WriteXml` de la classe `DataSet`.

Le gros du travail de la fonction `GenererFichierXml` consistera donc à charger les données de la commande (en-tête et lignes) dans un objet `DataSet`, à l'aide d'objets `SqlDataAdapter` comme nous avons pu le voir au chapitre précédent ; ceci étant fait, il suffira d'appeler la méthode `WriteXml` en lui passant en paramètre le nom du fichier à créer.

Afin de nous assurer que les lignes de commandes seront bien incluses à l'intérieur de l'élément commande dans le fichier XML, nous définirons une relation entre les tables `Commande` et `LigneCommande` du `DataSet`, dont il sera automatiquement tenu compte lors de la génération du fichier XML.

Fichier XmlGenerator.vb (Fonction GenererFichier Xml / Version VB.NET)

```
Public Sub GenererFichierXml(xmlFile As String)

  Dim myDataSet As DataSet
  Dim CommandAdapter,LinesAdapter As SqlDataAdapter
  myDataSet = new DataSet()

 CommandAdapter = new SqlDataAdapter("DetailsCommande",Connection)
 CommandAdapter.SelectCommand.CommandType = CommandType.StoredProcedure
 CommandAdapter.SelectCommand.Parameters.Add("@CommandeID",SqlDbType.Int)
 CommandAdapter.SelectCommand.Parameters("@CommandeID").Value = NoCommande
 CommandAdapter.Fill(myDataSet,"Commande")

 LinesAdapter = new SqlDataAdapter("ListeLignesCommandes",Connection)
 LinesAdapter.SelectCommand.CommandType = CommandType.StoredProcedure
 LinesAdapter.SelectCommand.Parameters.Add("@CommandeID",SqlDbType.Int)
 LinesAdapter.SelectCommand.Parameters("@CommandeID").Value = NoCommande
 LinesAdapter.Fill(myDataSet,"LigneCommande")

 myDataSet.Relations.Add ("Commande_LigneCommande", _
           myDataSet.Tables("Commande").Columns("ID_Commande"), _
           myDataSet.Tables("LigneCommande").Columns("ID_Commande"))

 myDataSet.Relations("Commande_LigneCommande").Nested = True




 myDataSet.WriteXml(Path+xmlFile)

End Sub
```

Allocation d'un objet `DataSet`

Chargement de l'en-tête de la commande dans la table `Commande` du `DataSet`, à l'aide de la procédure `DetailsCommande`, à laquelle on passe en paramètre la valeur de la variable membre NoCommande

Chargement des lignes de commande dans la table `LigneCommande` du `DataSet`, à l'aide de la procédure `ListeLignesCommandes`, à laquelle on passe en paramètre la valeur de la variable membre NoCommande

Définition d'une relation entre les tables `Commande` et `LigneCommande` du `DataSet`

Spécification de l'attribut `Nested` pour la relation, afin que les lignes de commandes soient incluses à l'intérieur de la commande dans le fichier XML généré

Génération du fichier XML

La version C# de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/cs/XmlGenerator.cs

Après compilation du fichier, testez les développements réalisés : à l'issue de la création d'une commande, vous êtes dorénavant redirigé vers la page récapitulative, qui permet d'afficher le fichier XML associé à la commande (figure 6-7).

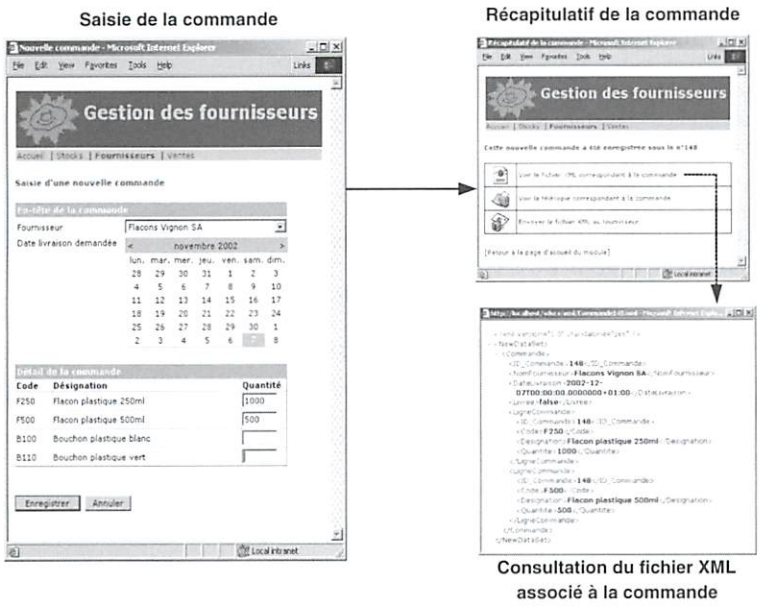


Figure 6–7 Consultation du fichier XML associé à la commande

RAPPEL Création d'un répertoire xml

N'oubliez pas de créer un répertoire `xml` situé sous la racine de l'application, dans lequel seront stockés les fichiers générés ; le compte sous lequel s'exécute le processus principal d'ASP.NET (`aspnet_wp.exe`) doit avoir accès à ce répertoire.

Lecture de données XML depuis une base SQL

Une alternative pour lire des données sous forme XML depuis une base SQL consiste à utiliser un objet `XmlTextReader`, obtenu grâce à la méthode `ExecuteXmlReader` de la classe `SqlCommand` qui permet à l'image de `SqlDataReader` de lire séquentiellement des données depuis la base vers une chaîne de caractères au format XML.

Génération d'une télécopie à l'aide d'une transformation XSL

Pour l'instant, nous avons uniquement fait appel aux fonctionnalités XML de la classe `DataSet`, qui ne constituent qu'une très petite partie des possibilités XML de la bibliothèque .NET : l'espace de nommage `System.Xml` et ses descendants hiérarchiques contiennent un grand nombre de classes qui permettent la création, la manipulation, la validation de documents XML, ainsi que la réalisation de transformations XSL, que nous allons illustrer plus particulièrement dans cette section.

Les fonctionnalités XSL reposent principalement sur la classe `XslTransform`, implémentée dans l'espace de nommage `System.Xml.Xsl`, qui permet de réaliser la transformation d'un document source représenté par un objet de type `XmlDocument` vers un flux de sortie, généré grâce à un objet `XmlTextWriter` (voir figure 6-8).

Avant tout, définissons une feuille de style XSL (nommée `GenererTelecopie.xsl`, à placer dans le répertoire `xml` sous la racine de l'application) qui va produire une télécopie au format HTML contenant le nom du fournisseur, son numéro de télécopie, le numéro de commande, ainsi que le détail des lignes de commandes.

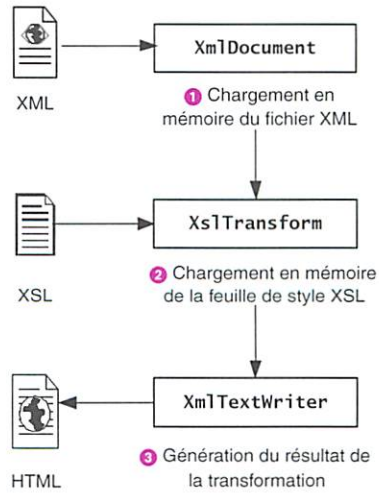


Figure 6–8 Application d'une transformation XSL

Fichier GenererTelecopie.xsl

On spécifie que la feuille de style produira, en sortie, un fichier HTML.

On affiche le nom du destinataire, son numéro de télécopie et l'objet du message (numéro de la commande).

On utilise une boucle `for-each` pour parcourir les lignes de commandes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
<xsl:output method="html"/>
<xsl:template match="/" >
<html>
<body>
   <h4>TELECOPIE</h4>
   <table>
   <tr><td><i>Expéditeur :</i></td><td>Ste Savons du Soleil</td></tr>
   <tr><td><i>Destinataire :</i></td>
   <td><xsl:value-of select="NewDataSet/Commande/NomFournisseur"/></td></tr>
   <tr><td><i>Télécopie :</i></td>
   <td><xsl:value-of select="NewDataSet/Commande/Telecopie"/></td></tr>
   <tr><td><i>Objet :</i></td>
   <td>Commande n°<xsl:value-ofselect="NewDataSet/Commande/ID_Commande"/></td>
   </tr>
   </table>
   <p><b>Détail de la commande:</b></p>
   <table border='1'>
      <tr>
         <td><b>Code</b></td>
         <td><b>Quantité</b></td>
      </tr>
      <xsl:for-each select="/NewDataSet/Commande/LigneCommande">
      <tr>
         <td><xsl:value-of select="./Code"/></td>
         <td><xsl:value-of select="./Quantite"/></td>
      </tr>
      </xsl:for-each>
   </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

ALLER PLUS VITE

Télécharger le fichier xsl

Cette feuille de style est téléchargeable sur le site de l'étude de cas :

- www.savonsdusoleil.com

Plus d'informations sur la syntaxe XSL

Si vous souhaitez obtenir plus d'informations sur la syntaxe et les possibilités des feuilles de styles XSL, nous vous conseillons de vous référer à un des ouvrages cités en référence au début de ce chapitre.

Il ne nous reste plus qu'à implémenter la fonction `GenererTelecopie` de notre classe `XmlGenerator` afin qu'elle réalise successivement :

- le chargement en mémoire du document XML à transformer grâce à la méthode `Load` de `XmlDocument` ;
- le chargement en mémoire de la feuille de style grâce à la méthode `Load` de `XslTransform` ;
- puis, pour finir, la réalisation de la transformation grâce à la méthode `Transform` de `XslTransform`, qui prend en paramètres le document source et un objet `XmlTextWriter` spécifiant les caractéristiques du fichier à produire (chemin, encodage, etc.).

Voici le code correspondant en VB.NET :

Fichier XmlGenerator.vb (Fonction GenererTelecopie/ Version VB.NET)

```
Public Sub GenererTelecopie(xml As String,xsl As String, output As String)

    Dim myTransform As XslTransform
    Dim myDocument  As XmlDocument
    Dim myWriter    As XmlTextWriter

    myDocument = new XmlDocument()
    myDocument.Load(Path+xmlFile)

    myTransform = new XslTransform()
    myTransform.Load(Path+xslFile)

    myWriter = new XmlTextWriter(Path+outputFile,Encoding.UTF8)
    myTransform.Transform(myDocument,Nothing,myWriter)
    myWriter.Close()
End Sub
```

- On passe en paramètre le nom du fichier XML source, le nom de la feuille de style XSL et le nom du fichier de sortie
- Déclaration des variables
- Chargement du fichier XML source
- Chargement de la feuille de style XSL
- Application de la transformation (on spécifie le mode d'encodage UTF8 pour le fichier de sortie)

La version C# de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/cs/XmlGenerator.cs

N'oubliez pas de compiler à nouveau la classe `XmlGenerator`, avant de tester l'application de la transformation, présentée figure 6-9.

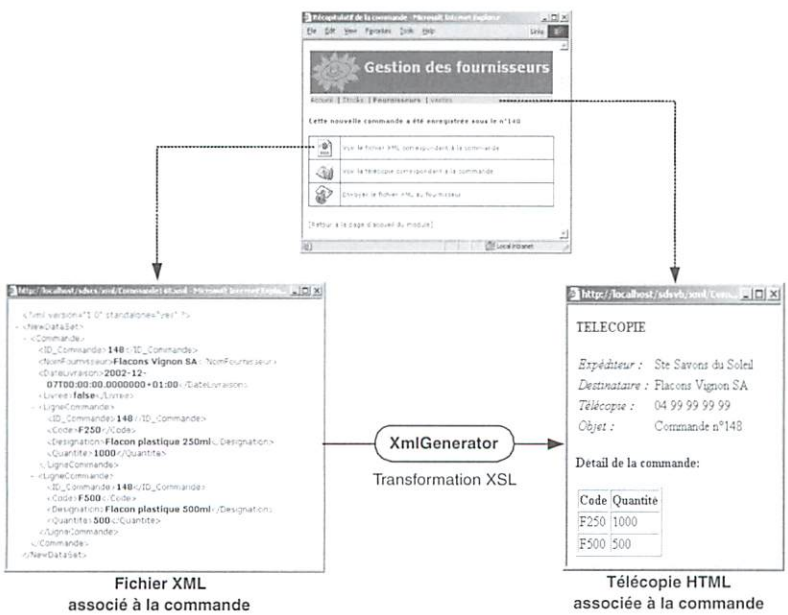


Figure 6–9 Consultation de la télécopie générée via une transformation XSL

Après avoir illustré ces quelques possibilités offertes par la bibliothèque .NET en matière de gestion des données XML, nous allons terminer par l'implémentation de l'envoi par messagerie du fichier XML associé à la commande.

RAPPEL Localisation du fichier xsl

Le fichier `GenererTelecopie.xsl` doit être placé dans le répertoire `xml` situé sous la racine de l'application.

Les autres possibilités XML de .NET

Nous n'avons cité que quelques-unes des fonctionnalités XML disponibles dans la bibliothèque .NET, laquelle permet également de valider des documents XML (`XmlValidatingReader`), d'utiliser XPath pour pointer vers des sections et des éléments spécifiques de documents XML (`XPathDocument`), ou encore de créer et manipuler des documents XML (`XmlDocument`).

Envoi du fichier XML par messagerie

ASP.NET intègre désormais en standard les outils nécessaires à l'envoi de fichiers par messagerie grâce aux classes de l'espace de nommage `System.Web.Mail` :

- la classe `MailMessage` répresente un message : expéditeur, destinaire(s), sujet, corps du message, format du message ;
- la classe `MailAttachement` représente une pièce jointe ;
- la classe `SmtpMail` permet d'effectuer l'envoi du message via un serveur SMTP.

L'envoi du message sera géré par une page ASP.NET simple (dans une fenêtre popup) qui demandera à l'utilisateur de saisir son adresse, effectuera l'envoi puis affichera le résultat de l'opération.

Commençons par réaliser la maquette de cette page :

1. Démarrez Web Matrix.
2. Créez une nouvelle page nommée `EnvoiCommande.aspx`.
3. Saisissez le titre « Simulation envoi commande ».
4. Saisissez le texte « Entrez votre adresse e-mail ».
5. À la droite de ce texte, insérez un contrôle serveur de type `TextBox` nommé `Adresse`.
6. Encore plus à droite, insérez un contrôle serveur de type `Button` nommé `Envoyer` et spécifiez la valeur « Envoyer » pour la propriété `Text`.
7. Créez un gestionnaire d'événement `Envoyer_Click` associé à l'événement `Click` du bouton `Envoyer`.
8. Sur la ligne du dessous, insérez un contrôle serveur de type `Label` nommé `Resultat`, qui sera utilisé pour afficher le résultat de l'envoi du message.
9. À côté, insérez un contrôle de validation de type `RegularExpressionValidator` nommé `ValiderAdresse`, qui sera utilisé pour vérifier la validité de l'adresse saisie ; paramétrez ce contrôle suivant les indications fournies dans le tableau :

Propriété	Valeur
`ControlToValidate`	Adresse
`ErrorMessage`	Merci de saisir une adresse e-mail valide
`ValidationExpression`	\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

10. Enfin, insérez en bas à droite un lien HTML simple nommé « Fermer cette fenêtre » et associez le script `window.close()` à l'événement client `onclick` de ce lien.

À propos de la gestion des messages

Les classes de gestion des messages d'ASP.NET sont basées sur les Collaboration Data Objects (CDO) de Windows, autrefois fournis uniquement avec Exchange Server.

À propos de ValidationExpression

Pour insérer automatiquement une expression régulière correspondant à une adresse de messagerie Internet, vous pouvez utiliser la boîte de dialogue associée au champ `ValidationExpression` de la feuille de propriétés du contrôle de validation (figure 6-10).

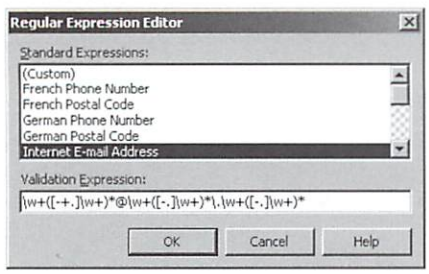


Figure 6–10 Insertion d'une expression régulière

La maquette correspondante est présentée figure 6-11 et le contenu HTML est spécifié ci-dessous (les contrôles serveur sont en couleur) :

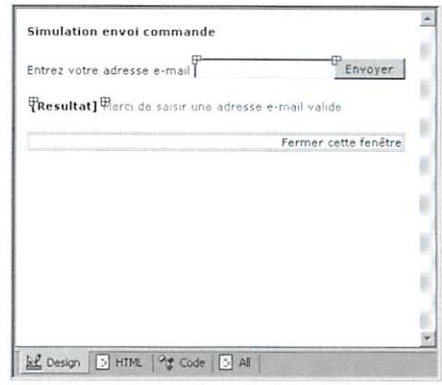


Figure 6–11 Maquette de la page d'envoi d'une commande

Fichier EnvoiCommande.aspx (Partie graphique)

```
<html>
<head>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <form runat="server">
    <h4>Simulation envoi commande</h4>
    <p>Entrez votre adresse e-mail
    <asp:TextBox id="Adresse" runat="server"/>
    <asp:Button id="Envoyer" onclick="Envoyer_Click"
        runat="server"  Text="Envoyer"/>
    </p><p>
    <asp:Label id="Resultat" runat="server" Font-Bold="True"/>
    <asp:RegularExpressionValidator id="ValiderAdresse"
        runat="server"
        ErrorMessage="Merci de saisir une adresse e-mail valide"
        ControlToValidate="Adresse"
        ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
    Merci de saisir une adresse e-mail valide
    </asp:RegularExpressionValidator>
   </p><p>
   <table width="370">
    <tr><td align="right">
      <a onclick="window.close()" href=".">Fermer cette fenêtre</a>
    </td></tr>
  </table>
</form>
</body>
</html>
```

Le code de la page sera entièrement contenu dans le gestionnaire d'événement `Envoyer_Click` et effectuera les opérations suivantes :

- récupération du numéro de commande, passé en paramètre sur la chaîne de requête ;
- création et paramétrage d'un objet `MailMessage` représentant le message à envoyer ;
- ajout d'une pièce jointe en utilisant un objet `MailAttachment` ;
- envoi du message avec `SmtpMail` ;
- affichage du résultat de l'envoi (réussite ou échec).

Avant d'implémenter le code, ajoutez en haut de la page la directive qui permet de faire référence aux classes de l'espace de nommage `System.Web.Mail` (en vous plaçant dans l'onglet All de Web Matrix) :

```
<%@ import Namespace="System.Web.Mail" %>
```

À propos de la gestion des exceptions

Dans le code de cette page, nous utilisons le mécanisme de gestion d'exceptions, qui permet d'intercepter et de traiter les erreurs ; nous aurons l'occasion d'en parler plus longuement dans le chapitre 9.

Fichier EnvoiCommande.aspx (Version VB.NET)

Récupération du numéro de commande

Allocation et initialisation d'un nouveau message (l'adresse du destinataire est récupérée dans le contrôle serveur Adresse)

Ajout, en pièce jointe, du fichier XML correspondant à la commande

Envoi du message (n'oubliez pas de spécifier le nom ou l'adresse de votre serveur SMTP). Notez que Smtp et Send sont des membres statiques, et qu'il n'est donc pas nécessaire d'instancier un objet SmtpMail

En cas d'erreur, on intercepte l'exception et on affiche le message correspondant

```
Sub Envoyer_Click(sender As Object, e As EventArgs)
  Try
    Dim id, virtualPath, physicalPath As String
    id = Request.Params("id")

    Dim mail As MailMessage
    mail = new MailMessage()
    mail.From = "webmaster@savonsdusoleil.com"
    mail.To = Adresse.Text
    mail.Subject = "Nouvelle commande"
    mail.Body = "Ci-joint le fichier XML correspondant à la commande"+ID
    mail.BodyFormat = MailFormat.Html

    virtualPath  = Request.ApplicationPath+"\\xml\\Commande"+id+".xml"
    physicalPath = Request.MapPath(VirtualPath)
    Dim myAttachment As MailAttachment
    myAttachment = new MailAttachment(physicalPath)
    mail.Attachments.Add(myAttachment)

    SmtpMail.SmtpServer = "VotreServeurSmtp"
    SmtpMail.Send(mail)
    Resultat.Text = "Le message a été correctement envoyé !"

  Catch myException As Exception
    Resultat.Text = "Echec de l'envoi du message :'"+myException.Message+"'"
  End Try
End Sub
```

La version C# de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/cs/EnvoiCommande.aspx

Serveur SMTP requis

Pour réaliser l'envoi du message, vous devez disposer d'un serveur SMTP (Simple Mail Transfer Protocol), dont il faut spécifier l'adresse (ou le nom, s'il peut être résolu) dans la propriété SmtpServer de SmtpMail.

Vous pouvez, au choix, utiliser un serveur SMTP externe (par exemple, celui de votre fournisseur d'accès Internet) ou installer un serveur en local sur la machine. Dans ce cas, une solution simple consiste à utiliser le serveur STMP fourni par défaut avec Windows, qui fait partie intégrante d'Internet Information Server :

1. Choisissez *Ajout/Suppression de Programmes* dans le panneau de configuration.
2. Choisissez *Installation de composants Windows*.
3. Choisissez *Internet Information Services (IIS)* et cliquez sur *Détails*.
4. Dans la fenêtre qui apparaît (voir figure 6-12), sélectionnez la case *SMTP Service* et cliquez sur OK.

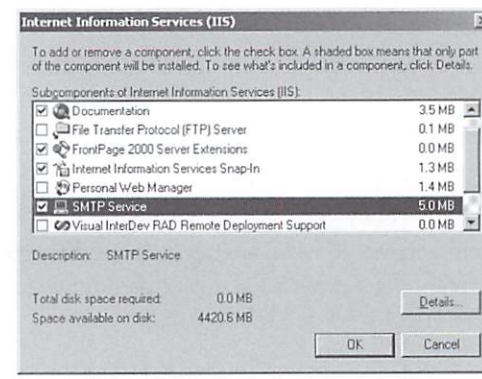


Figure 6–12 Installation du serveur SMTP d'IIS

Il reste une dernière opération à effectuer avant de pouvoir tester notre nouvelle page : paramétrer le contrôle `EnvoyerXML` de la page `FinCommande.aspx` afin qu'il pointe vers la page d'envoi du fichier, à laquelle il passera en paramètre l'identifiant de la commande.

Fichier FinCommande.aspx (Fin de la fonction Page_Load/Version VB.NET)

```
...
Dim MailPopupUrl,MailPopupArgs,MailUrl As String
MailPopupUrl  = "EnvoiCommande.aspx?id="+ID
MailPopupArgs = "'scrollbars=no,status=no,resizable=yes,width=400,height=150'"
MailUrl = "javascript:openpopup('"+MailPopupUrl+"',"+MailPopupArgs+")"
EnvoyerXML.NavigateURL=MailUrl
```

Le résultat de l'exécution est illustré figure 6-13.

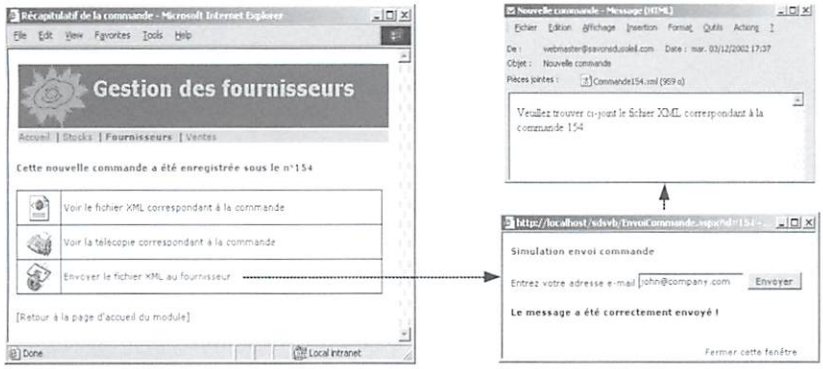


Figure 6–13 Envoi d'un message contenant le fichier associé à la commande

En résumé...

Dans ce chapitre, nous avons appris comment générer simplement un document XML à partir d'une structure de données contenue dans un objet `DataSet`, comment mettre en œuvre une transformation XSL à l'aide de la classe `XslTransform` et envoyer un message à l'aide des classes de l'espace de nommage `System.Web.Mail`.

Nous n'avons illustré que quelques-unes des nombreuses possibilités XML de la bibliothèque .NET, laquelle permet également de valider et manipuler des documents XML (ajout, suppression, modification de nœuds), de générer automatiquement des schémas XML ou encore d'utiliser XPath.

Au passage, nous avons décrit les techniques nécessaires à la création d'un objet métier encapsulant des fonctionnalités spécifiques, en soulignant l'intérêt d'une telle architecture.

Dans le prochain chapitre, nous allons développer un contrôle serveur spécifique qui sera utilisé dans le module de gestion des ventes de notre étude de cas.

Personnaliser l'ergonomie avec les contrôles serveur spécifiques

7

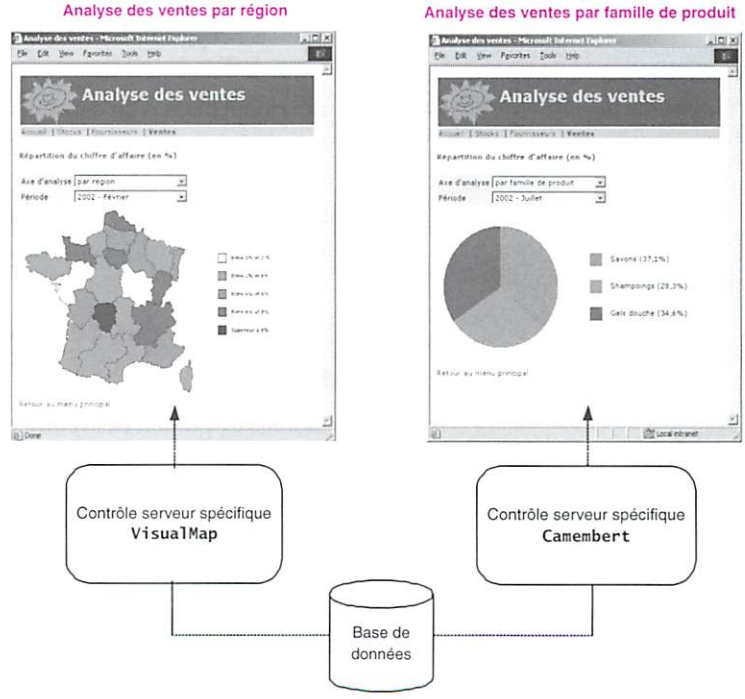
ASP.NET

Contrôle serveur spécifique | Global Assembly Cache (GAC) | WebControl | Render | System.Drawing

SOMMAIRE

MOTS-CLÉS

- Contrôle serveur spécifique
- Global Assembly Cache (GAC)
- WebControl
- Render
- System.Drawing



Dans ce chapitre, nous allons réaliser le module de gestion des ventes de notre étude de cas en faisant appel à des contrôles serveur spécifiques pour personnaliser l'interface : dans un premier temps, nous ferons appel à un contrôle du marché pour afficher la répartition des ventes par région sur une carte de France, puis, nous développerons un contrôle serveur spécifique de type « camembert » pour afficher la répartition des ventes par famille de produit.

Utilisation de contrôles serveur spécifiques

Dans les chapitres précédents, nous avons eu l'occasion de voir le rôle central que jouent les contrôles serveur dans le développement de pages ASP.NET : une simple instruction placée dans la partie graphique de la page permet de générer automatiquement un contenu HTML complexe adapté au navigateur, avec de nombreuses possibilités de paramétrages.

Néanmoins, nous n'avons pour l'instant fait appel qu'à des contrôles serveur prédéfinis de la bibliothèque ASP.NET qui, bien qu'elle soit très riche (plus de quarante contrôles dans l'espace de nommage `System.Web.UI.WebControls`), peut s'avérer insuffisante dans certaines situations. Heureusement, il est possible d'étendre les possibilités de cette bibliothèque en développant des contrôles serveur spécifiques, dont le comportement (code HTML généré, paramètres) est fixé par le développeur.

Nous allons illustrer ce mécanisme en deux temps :

- utilisation d'un contrôle serveur spécifique du marché (`VisualMap`) pour afficher les répartitions des ventes par région (génération dynamique d'une carte de France colorée en fonction des données contenues dans la base) ;
- développement d'un contrôle serveur spécifique simple pour afficher la réparation des ventes par famille de produit (génération dynamique d'un graphique composé de secteurs colorés)

Nous allons commencer par réaliser la page principale du module de gestion des ventes, qui permettra à l'utilisateur de saisir les paramètres d'analyses souhaités.

Contrôle serveur spécifique

Un contrôle serveur spécifique est un composant compilé réutilisable qui permet d'encapsuler la génération de code HTML et inclut des possibilités de paramétrage par l'utilisateur. Permettant d'étendre la liste des contrôles serveur standards fournis avec ASP.NET, les contrôles serveur spécifiques dérivent de classe `WebControl` ; ils peuvent soit spécialiser un contrôle standard existant, soit combiner les fonctionnalités de plusieurs contrôles (contrôles composites), soit être développés ex nihilo.

NE PAS CONFONDRE Contrôles serveur spécifiques et contrôles utilisateur

Attention : ne pas confondre les contrôles serveur spécifiques, qui sont des composants compilés, et les contrôles utilisateur, qui sont des morceaux de pages ASP.NET stockés dans un fichier portant l'extension `.ascx` (voir chapitre 3).

Consultation des résultats de ventes par région avec VisualMap

Le module de gestion des ventes de notre étude de cas doit permettre la consultation de la répartition du chiffre d'affaires (en pourcentage) pour un mois donné, suivant deux axes d'analyse : par famille de produit ou par région. Plutôt que d'afficher les résultats sous forme de tableaux de chiffres, comme nous l'avons fait au chapitre 4 pour la consultation des stocks à l'aide du contrôle `DataGrid`, nous souhaitons afficher les résultats sous forme graphique, plus convaincante et plus facile à lire : nous allons utiliser pour cela des contrôles serveur spécifiques.

Nous allons commencer par réaliser la maquette de la page principale du module, qui permettra à l'utilisateur de choisir l'axe d'analyse (famille de produit ou région) et la période concernée ; puis nous inclurons dans cette page un contrôle serveur spécifique du marché, `VisualMap`, afin de consulter les résultats de ventes par région à travers un graphique.

Réalisation de la maquette de la page de consultation des ventes

En dehors du recours à un contrôle serveur spécifique, la page d'analyse des ventes ne fera appel qu'à des techniques déjà connues (voir figure 7-1) :

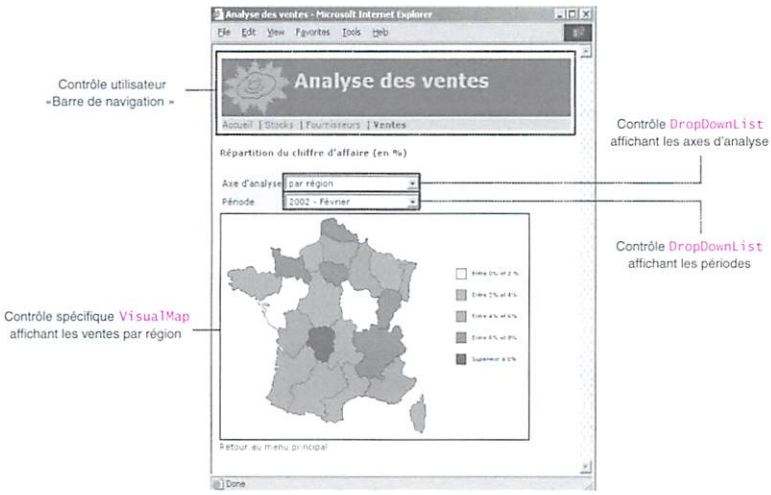


Figure 7–1 La page d'analyse des ventes par région

- utilisation de contrôles de type `DropDownList` pour afficher la liste des axes et des périodes d'analyse ;
- récupération des données de vente dans la base à l'aide d'objets `SqlCommand` et `SqlDataReader` ;
- recours au mécanisme des événements serveur pour implémenter la mise à jour automatique de la page en cas de modification des paramètres d'analyse.

Pour réaliser la maquette de cette page :

1. Démarrez Web Matrix.
2. Ouvrez le fichier `Ventes.aspx` créé au chapitre 3.
3. Insérez un titre « Répartition du chiffre d'affaires (en %) ».
4. Insérez un tableau HTML comportant deux lignes et deux colonnes.
5. Dans la première ligne, saisissez le texte « Axe d'analyse » dans la colonne de gauche et insérez un contrôle serveur nommé `ListeAxes` de type `DropDownList` dans la colonne de droite.
6. Dans la seconde ligne, saisissez le texte « Période » dans la colonne de gauche et insérez un contrôle serveur nommé `ListeMois` de type `DropDownList` dans la colonne de droite.
7. Afin de pouvoir réaliser le rechargement automatique de la page en cas de modification des paramètres d'analyse, positionnez à `True` la valeur de la propriété `AutoPostBack` pour ces deux contrôles et associez-leur les gestionnaires d'événements suivants :

Contrôle	Événement	Gestionnaire
ListeAxes	OnSelectedIndexChanged	ListeAxes_SelectedIndexChanged
ListeMois	OnSelectedIndexChanged	ListeMois_SelectedIndexChanged

8 Enfin, insérez en bas de la page un lien hypertexte simple : « Retour au menu principal », vers la page `default.aspx`

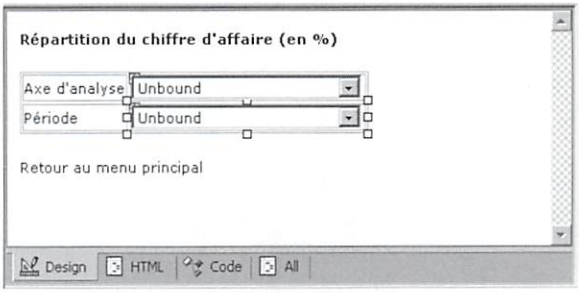


Figure 7–2 Maquette de la page d'analyse des ventes

La maquette correspondante est présentée figure 7-2 et le contenu HTML est spécifié ci-après (les contrôles serveur sont en couleur) :

Fichier Ventes.aspx (Partie graphique)

```
<html>
<head>
    <title>Analyse des ventes</title>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
 <SDS:NavBar id="MyNavBar" runat="server" SelectedIndex="3">
 </SDS:NavBar>
 <form runat="server">
  <h4>Répartition du chiffre d'affaire (en %)</h4>
  <table bgcolor="#ffffc0">
    <tr><td>Axe d'analyse</td>
    <td><asp:DropDownList id="ListeAxes" runat="server"
           OnSelectedIndexChanged="ListeAxes_SelectedIndexChanged"
           AutoPostBack="True"/>
    </td></tr>
    <tr><td>Période</td>
    <td><asp:DropDownList id="ListeMois" runat="server"
           OnSelectedIndexChanged="ListeMois_SelectedIndexChanged"
           AutoPostBack="True"/>
    </td></tr>
  </table>
  <p></p>
  <a href="default.aspx">Retour au menu principal</a>
 </form>
</body>
</html>
```

Il est maintenant temps de s'occuper de l'élément central de la page : le contrôle serveur spécifique `VisualMap` qui va nous permettre d'afficher les résultats de vente par région sous forme graphique.

Téléchargement et installation du contrôle serveur spécifique VisualMap

À l'image de ce qui existait pour les composants COM/ActiveX, de nombreux contrôles serveur spécifiques sont disponibles sur le marché, en complément des contrôles serveur standards fournis avec ASP.NET : génération de codes-barres, calendriers complexes, affichages industriels (jauges, compteurs), génération de feuilles de calcul de type tableur ou de documents PDF, barres de menus Web, affichages arborescents, générations dynamiques d'images liées à des données, pour n'en citer que quelques-uns.

Ces composants sont généralement téléchargeables sous la forme d'une version d'évaluation pleinement fonctionnelle, valable pendant trente jours : c'est le cas du composant `VisualMap` que nous allons utiliser pour notre étude de cas, téléchargeable depuis le site www.visualmap.net.

Pour installer le composant `VisualMap` sur votre poste de développement :

1. Téléchargez la version d'évaluation de VisualMap (France) depuis www.visualmap.net.
2. Exécutez le programme d'installation (voir figure 7-3). C'est tout !

Pour pouvoir faire facilement référence au composant, nous allons l'installer dans la barre d'outils Custom Controls de Web Matrix :

1. Dans Web Matrix, activez la barre d'outils Custom Controls.
2. Cliquez avec le bouton droit de la souris et sélectionnez l'option Add Local Toolbox Components.
3. Dans la boîte de dialogue qui apparaît (voir figure 7-5), sélectionnez l'assemblage `Topic.Web.VisualMap.France`, qui contient le contrôle venant d'être installé, ajoutez-le à la liste des assemblages sélectionnés en cliquant sur le bouton Add, puis cliquez sur OK.
4. Le contrôle `VisualMap` apparaît maintenant dans la barre d'outils Custom Controls de Web Matrix (voir figure 7-6).

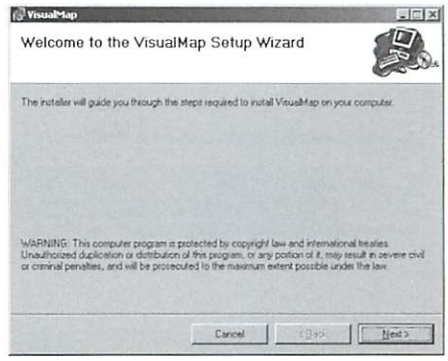


Figure 7–3 Installation de VisualMap

Où le composant est-il installé ?

Lorsqu'on installe un contrôle serveur spécifique sur une machine de développement, il est généralement installé, au sein d'un assemblage, dans le Global Assembly Cache (GAC), ce qui le rend disponible à toutes les applications Web de la machine. Une copie de l'assemblage, accompagnée de fichiers d'aide et d'exemples, est généralement placée dans le répertoire `Program Files`.

Où trouver des contrôles serveur spécifiques ?

Une liste relativement exhaustive des composants .NET existants est disponible sur le site :

- www.componentsource.com

D'autre part, signalons la présence dans Web Matrix d'un moteur de recherche de composants en ligne (Online Component Gallery) basé sur l'interrogation d'un service Web, malheureusement associé à un catalogue peu fourni pour l'instant :

1. Dans Web Matrix, activez la barre d'outils *Custom Controls*.
2. Cliquez avec le bouton droit de la souris et sélectionnez l'option *Add Online Toolbox Components*.
3. Une boîte de dialogue qui permet de rechercher des composants par catégorie ou par mots-clés apparaît (voir figure 7-4).

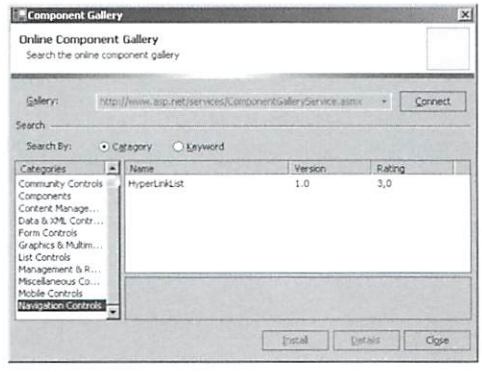


Figure 7–4 Recherche de composants en ligne avec Web Matrix

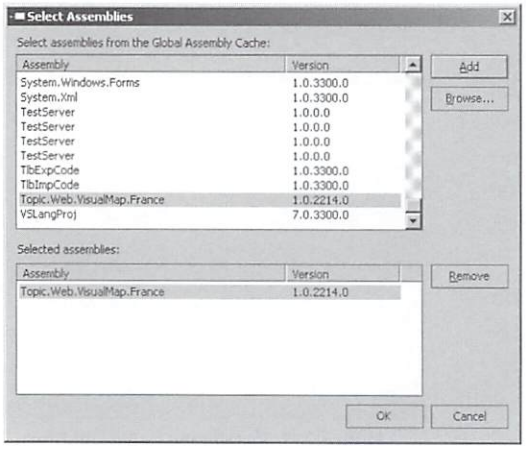


Figure 7–5
Ajout du composant VisualMap à la barre d'outils de Web Matrix

Figure 7–6
La barre d'outils Custom Controls de Web Matrix

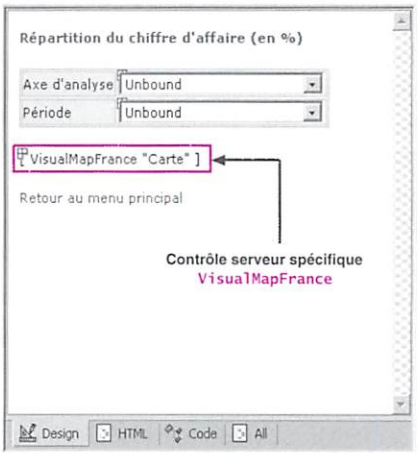


Figure 7–7
Maquette de la page d'analyse des ventes après insertion du composant `VisualMap`

Le contrôle serveur spécifique étant installé, nous allons maintenant l'intégrer à notre page d'analyse des résultats de vente, puis réaliser son paramétrage.

Intégration du contrôle VisualMap dans la page d'analyse des ventes

Pour insérer un contrôle de type `VisualMapFrance` dans la page d'analyse des ventes (voir figure 7-7) :

1. Placez-vous dans l'onglet Design de Web Matrix.
2. Activez la barre d'outils Custom Controls.
3. Sélectionnez le contrôle `VisualMapFrance` et faites-le glisser vers la zone d'édition.
4. Donnez au contrôle le nom `Carte`, en utilisant la feuille de propriétés associée.

En basculant dans l'onglet HTML, on constate que Web Matrix a automatiquement inséré une balise faisant référence au contrôle :

```
<n0:VisualMapFrance id="Carte" runat="server"></n0:VisualMapFrance>
```

où `n0` est un préfixe arbitrairement choisi par l'environnement de développement.

En basculant maintenant dans l'onglet All, on constate qu'une directive `<%@ Register %>` a été ajoutée, laquelle permet, entre autres, de spécifier dans quel assemblage se trouve le contrôle :

```
<%@ Register TagPrefix="n0" Namespace="Topic.Web.VisualMap.France"
      Assembly="Topic.Web.VisualMap.France, Version=1.0.2214.0,
      PublicKeyToken=615c61a4eeee5b74" %>
```

où :

- `TagPrefix` spécifie le préfixe qui permettra de faire référence au contrôle depuis le contenu HTML (ce préfixe peut être modifié par l'utilisateur, à condition que celui-ci modifie également la valeur correspondante dans l'onglet HTML) ;
- `Namespace` indique l'espace de nommage dans lequel se trouve le contrôle ;
- `Assembly` indique le nom de l'assemblage au sein duquel est stocké le contrôle (rappelons qu'un assemblage peut contenir plusieurs espaces de nommage) ;
- `Version` indique le numéro de la version souhaitée (plusieurs versions d'un même contrôle peuvent cohabiter sur une machine) ;
- `PublicKeyToken` indique la clé publique avec laquelle a été signé l'assemblage du contrôle (tout composant installé dans le Global Assembly Cache doit être signé).

Paramètrage du contrôle VisualMap pour réaliser l'affichage des ventes par région

Le contrôle `VisualMapFrance` permet d'afficher des valeurs numériques liées à des éléments géographiques (régions, départements) sous la forme d'une image générée dynamiquement.

À l'image des contrôles serveur liés aux données que nous avons déjà rencontrés (`DataGrid`, `DropDownList`, `Repeater`...), il comporte une propriété `DataSource` qui permet de spécifier la source de données associée, ainsi que deux propriétés `DataValueField` et `DataKeyField` qui permettent respectivement d'indiquer le nom du champ contenant la valeur à représenter et celui du champ correspondant à l'identifiant géographique (numéro de la région ou du département), comme l'illustre la figure 7-8.

Par conséquent, l'implémentation du code correspondant va être assez similaire à celle réalisée au chapitre 4 pour le module de suivi des stocks :

- lors du premier chargement de la page (réperé grâce à la propriété `IsPostBack`) : remplissage des listes déroulantes (grâce à deux fonctions utilitaires `RemplirListeAxesAnalyse` et `RemplirListeMois`) et paramétrage du contrôle serveur `VisualMapFrance` (format, source de données) ;
- lorsque l'utilisateur sélectionne un nouveau mois dans la liste déroulante `ListeMois` : modification de la source de données associée au contrôle serveur `VisualMapFrance` (récupération des ventes par région pour le nouveau mois sélectionné).

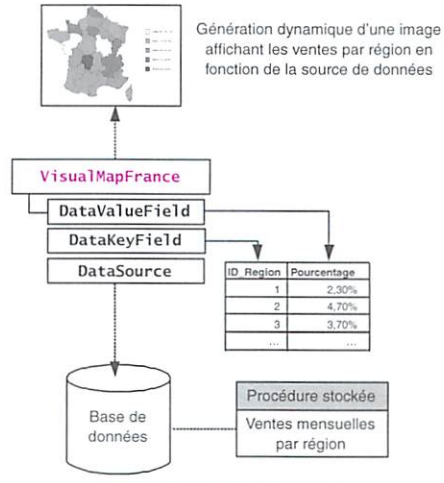


Figure 7–8 Architecture du contrôle serveur `VisualMapFrance`

Fichier Ventes.aspx (Version C#)

Cette fonction utilitaire remplit la liste des axes d'analyse disponibles, en associant au contrôle `ListeAxes`, de type `DropDownList`, une source de données en mémoire représentée par un objet `ArrayList`.

```
void RemplirListeAxesAnalyse()
{
    ArrayList list = new ArrayList();
    list.Add("par région");
    list.Add("par famille de produit");
    ListeAxes.DataSource = list;
    ListeAxes.DataBind();
}
```

Cette fonction utilitaire remplit le contrôle `ListeMois` (on se limite à l'année 2002, pour simplifier), en utilisant la même technique que dans la fonction précédente.

```
void RemplirListeMois()
{
    ArrayList list = new ArrayList();
    list.Add("2002 - Janvier");
    list.Add("2002 - Février");
    list.Add("2002 - Mars");
    list.Add("2002 - Avril");
    list.Add("2002 - Mai");
    list.Add("2002 - Juin");
    list.Add("2002 - Juillet");
    list.Add("2002 - Août");
    list.Add("2002 - Septembre");
    list.Add("2002 - Octobre");
    list.Add("2002 - Novembre");
    list.Add("2002 - Décembre");
    ListeMois.DataSource = list;
    ListeMois.DataBind();
}
```

Lors du premier chargement de la page (autrement dit, lorsque `IsPostBack` vaut `false`), on remplit les listes déroulantes, grâce aux fonctions utilitaires déclarées plus haut, puis on initialise le contrôle `VisualMapFrance` en spécifiant le mode d'affichage (`Regions`), le type des données représentées (`Percentage`), la définition des niveaux de couleurs à utiliser (génération automatique), ainsi que les noms des champs de la source de données correspondant à la valeur à représenter (`DataValueField`) et à l'identifiant géographique (`DataKeyField`) ; enfin, on appelle la fonction `ParametrerGraphique`, qui va associer la source de données adéquate au contrôle.

```
void Page_Load(Object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        RemplirListeAxesAnalyse();
        RemplirListeMois();
        Carte.DisplayMode = VisualMapFrance.DisplayModeEnum.Regions;
        Carte.ValueType   = VisualMapFrance.ValueTypeEnum.Percentage;
        Carte.AutoGenerateLevels = true;
        Carte.DataValueField = "Pourcentage";
        Carte.DataKeyField = "ID_Region";
        ParametrerGraphique();
    }
}
```

Cette fonction initialise la source de données du contrôle `VisualMapFrance` avec la liste des ventes par région correspondant au mois sélectionné (le travail est sous-traité à la fonction `GetDataSource_Regions`).

```
void ParametrerGraphique()
{
    int RegionID = ListeMois.SelectedIndex + 1;
    Carte.DataSource = GetDataSource_Regions(RegionID);
}
```

Cette fonction récupère une source de données de type `DataView` résultat de l'exécution de la procédure `VentesMensuellesParRegion` à laquelle on passe en paramètre le numéro du mois (par exemple : « 200207 »), formaté grâce à la méthode `String.Format`.

```
DataView GetDataSource_Regions(int mois)
{
    string param = String.Format("2002{0:D2}",mois);
    string sql = "VentesMensuellesParRegion"
```

```
    SqlConnection myConnection = (SqlConnection)Session["myConnection"];
    SqlDataAdapter myAdapter = new SqlDataAdapter(sql,myConnection);
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
    myAdapter.SelectCommand.Parameters.Add("@Mois",SqlDbType.VarChar);
    myAdapter.SelectCommand.Parameters["@Mois"].Value = param;
    DataTable myDataTable = new DataTable();
    myAdapter.Fill(myDataTable);
    return myDataTable.DefaultView;
}

void ListeMois_SelectedIndexChanged(Object sender, EventArgs e)
{
   ParametrerGraphique();
}

void ListeAxes_SelectedIndexChanged(Object sender, EventArgs e)
{
   ParametrerGraphique();
}
```

Pour cela, on utilise les objets intermédiaires `SqlConnection`, `SqlDataAdapter` et `DataTable` comme au chapitre 4.

Ce gestionnaire d'événement est exécuté lorsque la page est rechargée suite à un changement de sélection effectué par l'utilisateur dans la liste des mois : il associe au contrôle `VisualMapFrance` la source de données associée au nouveau mois sélectionné (le travail est sous-traité à la fonction `Parametrer Graphique`).

Ce gestionnaire d'événement est exécuté lorsque la page est rechargée suite à un changement de sélection effectué par l'utilisateur dans la liste des axes d'analyse : pour l'instant, il est sans effet.

La version VB.NET de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/vb/Ventes.aspx

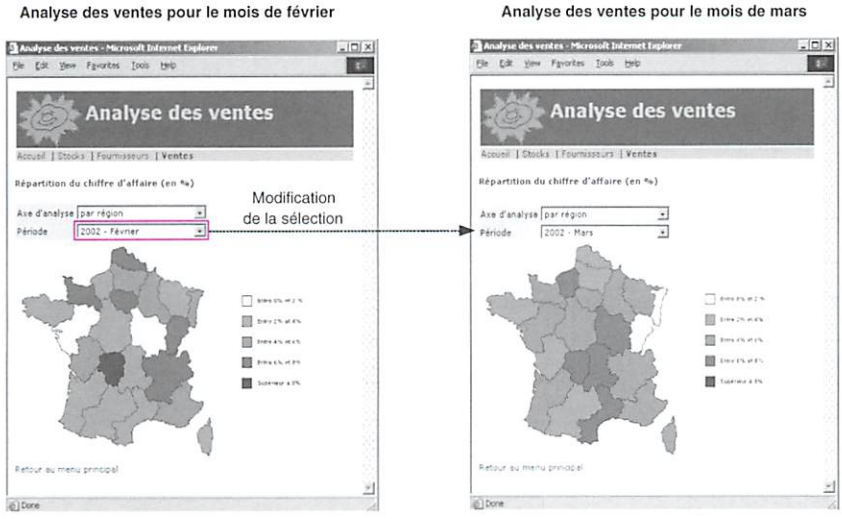


Figure 7–9 Cinématique de la page d'analyse des ventes par régions

Le résultat de l'exécution (présenté figure 7-9) est déjà appréciable en terme d'ergonomie, mais nous allons faire mieux : développer notre propre contrôle serveur spécifique, qui représentera la répartition du chiffre d'affaires par famille de produit sous la forme d'un graphique de type camembert, composé de secteurs colorés.

Création d'un contrôle serveur spécifique

Nous allons commencer par présenter le mécanisme de création d'un contrôle serveur spécifique, basé sur l'utilisation de l'héritage, puis nous implémenterons le contrôle `Camembert`, capable d'afficher des valeurs issues d'une source de données sous forme d'un graphique composé de secteurs, que nous intégrerons à notre page d'analyse des ventes.

Mécanisme de création d'un contrôle serveur spécifique

On pourrait presque dire, en simplifiant à peine, qu'il suffit de créer une classe dérivée de la classe `WebControl`, définie dans l'espace de nommage `System.Web.UI.WebControls`, pour obtenir un contrôle serveur spécifique fonctionnel.

En effet, la nouvelle classe ainsi définie hérite des principales fonctionnalités nécessaires à un contrôle serveur (par exemple, les propriétés `ID`, `Visible`, `BackColor` et `CssClass`, les méthodes `DataBind` et `FindControl`, ou encore les gestionnaires d'événements `Load` et `Unload`), lesquelles sont implémentées dans la classe `WebControl` (et ses parents), qui sert d'ailleurs de classe de base à l'ensemble des contrôles serveur standards (voir figure 7-10).

En réalité, il faut effectuer un petit travail supplémentaire en plus de l'héritage : spécifier le contenu HTML produit par le contrôle en spécialisant la méthode `Render`.

Héritage

L'héritage est un mécanisme utilisé dans le développement objet, permettant de définir une nouvelle classe non pas « à partir de zéro », mais à partir d'une classe existante, dite classe de base, à laquelle on ajoute des fonctionnalités ou dont on spécialise le comportement (la nouvelle classe est dite « dérivée » de la classe de base). L'intérêt principal de ce mécanisme est le gain de productivité dans le développement.

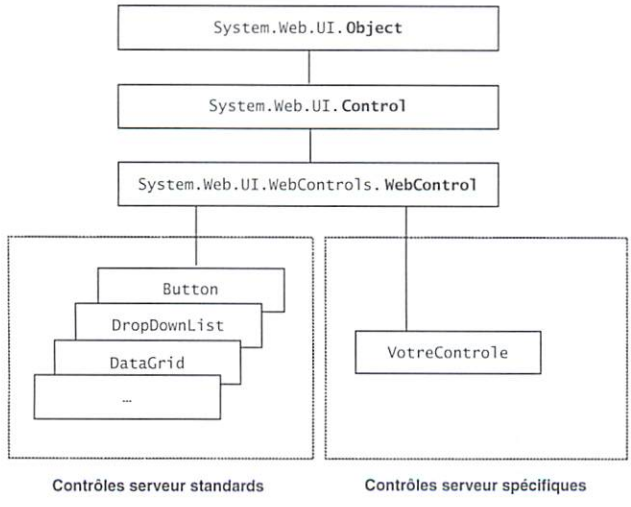


Figure 7–10 Hiérarchies de classes utilisées pour les contrôles serveur

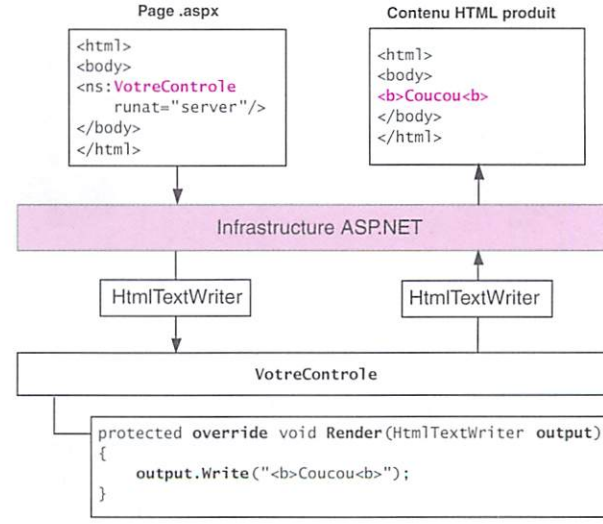


Figure 7–11 Architecture interne d'un contrôle serveur

Personnaliser le contenu HTML produit par le contrôle avec la méthode Render

Un contrôle serveur produit du contenu HTML ; pour implémenter un contrôle serveur spécifique, il faut spécifier le contenu qu'il va produire lors de son exécution : ceci s'effectue en redéfinissant la méthode `Render` du contrôle, qui prend en paramètre un objet de type `HtmlTextWriter` représentant le contenu HTML associé au contrôle (voir figure 7-11).

Nous allons mettre en œuvre ce mécanisme pour implémenter notre contrôle serveur spécifique `Camembert`, pour lequel la méthode `Render` effectuera la génération dynamique d'une image représentant un graphique divisés en secteurs, puis insèrera une balise `<img>` pointant vers cette image dans le contenu HTML produit par le contrôle.

Architecture du contrôle Camembert

Comme pour le contrôle `VisualMap` utilisé au début de ce chapitre, le contrôle `Camembert` reposera principalement sur la génération dynamique d'une image (graphique divisé en secteurs, accompagné d'une légende). Une fois l'image créée, le contrôle générera, en guise de contenu HTML, une balise `<img>` lui faisant référence (voir figure 7-12).

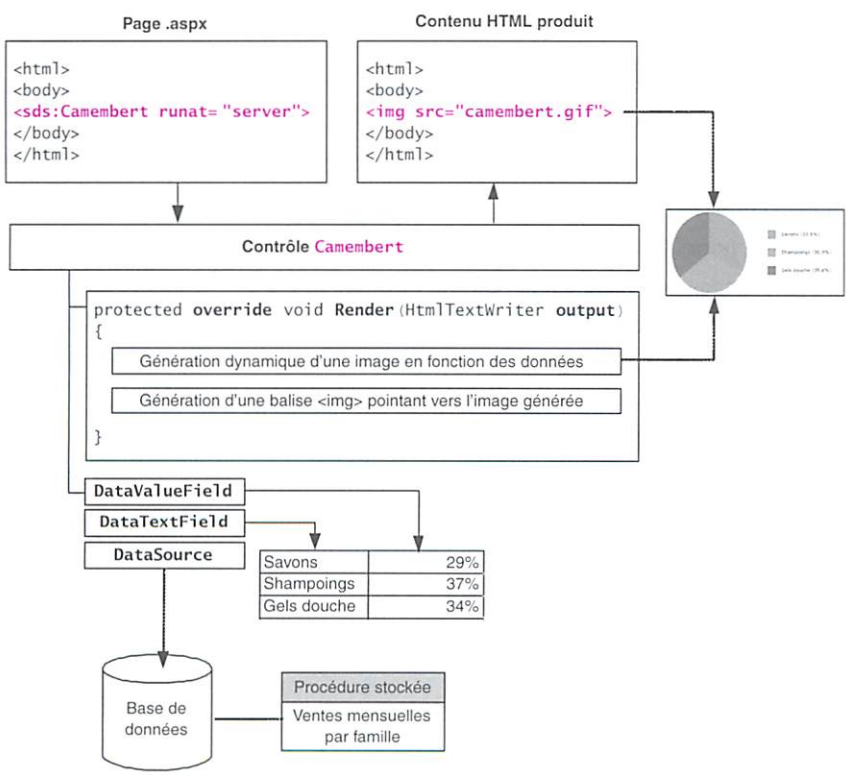


Figure 7–12 Architecture du contrôle `Camembert`

Le contrôle nécessitera trois propriétés pour paramétrer l'accès aux données :

- la propriété `DataSource` permettra de spécifier la source de données ;
- la propriété `DataValueField` spécifiera le nom du champ de la source de données correspondant à la valeur à afficher ;
- la propriété `DataTextField` spécifiera le nom du champ de la source de données correspondant à l'étiquette qui sera utilisée pour la légende.

Ces deux dernières propriétés seront stockées dans la propriété `ViewState` du contrôle (héritée de la classe `Control`), afin qu'elles conservent leurs valeurs lors d'un rechargement de la page suite au déclenchement d'un événement serveur.

La génération dynamique de l'image, qui peut apparaître comme la principale difficulté, sera en réalité assez facile à réaliser grâce aux puissantes fonctionnalités disponibles dans l'espace de nommage `System.Drawing`.

Dans un premier temps, nous allons réaliser l'implémentation du squelette de la classe `Camembert`, puis nous implémenterons la méthode `Render`, qui réalisera la partie principale du travail.

Création du contrôle Camembert

D'un point de vue pratique, un contrôle serveur spécifique est implémenté sous la forme d'une classe indépendante, dont nous allons créer simplement le squelette, à l'image de ce que nous avons fait au chapitre précédent pour créer un objet métier :

1. Démarrez Web Matrix.
2. Choisissez New dans le menu File.
3. Sélectionnez le type de fichier Class et spécifiez :
   - l'emplacement du nouveau fichier (le répertoire de votre application) ;
   - son nom (par exemple : Camembert) ;
   - le langage de votre choix (C# ou VB.NET) ;
   - le nom de la classe (par exemple : `Camembert`) ;
   - l'espace de nommage dans lequel sera stockée la classe (par exemple : `SDS.Controls`, pour « Savons du Soleil /Contrôles»).

Voici le code correspondant au squelette de la classe :

Fichier Camembert.cs (Version C#)

On importe tous les espaces de nommage dont on aura besoin pour l'implémentation de la classe

```
namespace SDS.Controls
{

    using System;
    using System.Data;
    using System.Web;
    using System.Web.UI;
    using System.Web.UI WebControls;
    using System.Drawing;
    using System.Drawing.Imaging;
```

```
    public class Camembert : WebControl
    {

        DataView _dataSource;


        public DataView DataSource
        {
            get {return _dataSource;  }
            set {_dataSource = value; }
        }
        public string DataValueField
        {
            get {return (string)ViewState["DataValueField"];  }
            set {ViewState["DataValueField"] = value; }
        }
        public string DataTextField
        {
            get {return (string)ViewState["DataTextField"];  }
            set {ViewState["DataTextField"] = value; }
        }
        protected override void Render(HtmlTextWriter output)
        {
           // Sera implémentée plus tard
        }
    }
}
```

- On spécifie que la classe `Camembert` dérive de la classe `WebControl`
- Variable membre privée utilisée pour le stockage de la source de données
- Propriété publique qui permet de spécifier la source de données
- Propriété publique qui permet de spécifier le nom du champ contenant la valeur à afficher (stockée dans `ViewState`)
- Propriété publique qui permet de spécifier le nom du champ contenant l'étiquette de texte associée à la valeur (stockée dans `ViewState`)
- Méthode `Render` (sera implémentée plus tard) ; le mot-clé `override` spécifie qu'il s'agit de la redéfinition d'une méthode de la classe de base

La version VB.NET de ce code est disponible en ligne à l'adresse :

› http://www.savonsdusoleil.com/src/vb/Camembert.vb

Même si ce contrôle n'effectue pas grand chose pour l'instant, vous pouvez néanmoins effectuer sa compilation, en utilisant la même syntaxe que celle détaillée dans le chapitre précédent.

Compilation de la version C# du contrôle

```
csc /out:bin\SDS.dll /t:library Camembert.cs /r:System.dll
  ➥ /r:System.Data.dll /r:System.Drawing.dll /r:System.Web.dll
```

Compilation de la version VB.NET du contrôle

```
vbc /out:bin\SDS.dll /t:library Camembert.vb /r:System.dll
  ➥ /r:System.Data.dll /r:System.Drawing.dll /r:System.Web.dll
```

Passons maintenant à l'implémentation de la méthode `Render`, qui constitue le cœur de notre contrôle serveur spécifique.

RAPPEL Répertoire bin

Le contrôle étant compilé sous la forme d'un assemblage privé, vous devez, si ce n'est pas encore fait, créer un répertoire `bin` sous la racine de votre application, dans lequel sera placé le fichier résultant de la compilation.

Implémentation de la génération de l'image au sein de la méthode Render

La méthode `Render`, dont le rôle est de spécifier le contenu HTML produit par le contrôle, fonctionnera en trois temps :

- génération dynamique du graphique camembert et de la légende associée à partir des valeurs de la source de données, en utilisant les classes `Bitmap` et `Graphics` de l'espace de nommage `System.Drawing` (notamment les méthodes `FillPie`, `FillRectangle` et `DrawString`) ;
- sauvegarde de l'image dans un fichier temporaire ;
- spécification d'un contenu HTML pointant vers le fichier généré (on utilisera une balise `<img>` faisant référence à l'image via une page intermédiaire, afin de contourner le problème de la conservation éventuelle de l'image dans le cache du navigateur client).

Fichier Camembert.cs (Méthode Render / Version C#)

```
protected override void Render(HtmlTextWriter output)
{
    Bitmap bm = new Bitmap(400,200);
    Graphics g = Graphics.FromImage(bm);

    g.Clear(Color.White);

    Font f = new Font("Verdana", 8);


    float startAngle = -90;
    float sweepAngle = 0;
    int i = 0;
    foreach(DataRowView dr in DataSource)
    {
        float  pourcentage = Convert.ToSingle(dr[DataValueField]);
        string etiquette   = dr[DataTextField].ToString();
       string legend = String.Format("{0} ({1:F1}%)",etiquette,pourcentage);
        sweepAngle = pourcentage*360/100;
        g.FillPie(GetBrush(i),10,10,180,180,startAngle,sweepAngle);
        startAngle += sweepAngle;
        g.FillRectangle(GetBrush(i),240,50+i*40,20,20);
        g.DrawString(legend, f, Brushes.Black,270,52+i*40);
        i++;
    }

      HttpRequest r = Context.Request;
      string path = r.MapPath(r.ApplicationPath)+"\\img";
      bm.Save(path+"\\"+"Camembert.gif",ImageFormat.Gif);

      output.Write("<img src='LoadImage.aspx?path="+
        ➥  path+"&src=camembert.gif'>");
```

Création d'une image bitmap de 400 x 200 pixels et obtention de l'objet `Graphics` qui lui est associé

Initialisation de l'image (tout en blanc)

Définition de la police qui sera utilisée pour la légende

Définition des variables nécessaires à la boucle de génération des secteurs angulaires : `startAngle` désigne l'angle de départ et `sweepAngle` désigne l'angle couvert par un secteur

Boucle principale qui récupère, pour chaque ligne de la source de données, la valeur à représenter et le texte correspondant pour la légende, dessine le secteur graphique correspondant avec `FillPie` (la couleur à utiliser en fonction du secteur est obtenue avec la fonction `GetBrush`, décrite plus loin), puis ajoute la légende sous la forme d'un rectangle de couleur (`FillRectangle`) suivi d'un texte (`DrawString`)

Sauvegarde de l'image vers un fichier nommé `Camembert.gif`, dans le sous-répertoire `img` de l'application (on utilise la propriété `Context` pour accéder à l'objet ASP.NET `Request`)

Spécification du contenu HTML produit par le contrôle (génération d'une balise `<img>` pointant indirectement vers l'image, via un fichier intermédiaire `LoadImage` dont le rôle est décrit dans le paragraphe suivant)

```
        f.Dispose();
        g.Dispose();
        bm.Dispose();
    }
    private Brush GetBrush(int index)
    {
        Brush br = Brushes.Black;
        switch(index)
        {
            case 0 :
                br = Brushes.CornflowerBlue;
            break;
            case 1 :
                br = Brushes.HotPink;
            break;
            case 2 :
                br = Brushes.MediumSpringGreen;
            break;
        }
        return br;
    }
```

Libération des ressources graphiques (qui, contrairement aux objets alloués en mémoire, ne sont pas automatiquement libérées par la CLR)

Cette fonction utilitaire renvoie des « pinceaux » dont la couleur diffère en fonction de l'index passé en paramètre (pour simplifier, on ne gère que trois niveaux de pinceaux)

La version VB.NET de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/vb/Camembert.vb

Avant de pouvoir utiliser le contrôle, il nous reste à implémenter la page `LoadImage.aspx`, utilisée dans le chargement de l'image, et dont nous allons à présent détailler le rôle et le fonctionnement.

Éviter les problèmes de maintien en cache côté client avec la page LoadImage

Le contrôle `Camembert` génère dynamiquement une image, stockée dans un fichier nommé `Camembert.gif` placé dans le répertoire `img` sous la racine de l'application.

La solution la plus simple pour faire référence à cette image depuis la page HTML générée est d'utiliser une balise `<img>` sous sa forme la plus simple :

```
<img src='./img/camembert.gif'>");
```

Néanmoins, cette solution présente un inconvénient important : l'image risque d'être conservée en mémoire dans le cache du navigateur côté client, ce qui interdirait d'obtenir le comportement souhaité pour notre contrôle (l'image restant inchangée après une modification du mois sélectionné).

Pour contourner ce problème, nous allons faire appel à une page intermédiaire qui prendra en paramètre, sur la ligne de requête, le chemin et le nom du fichier image à charger et produira en sortie l'image correspondante :

```
<img src='LoadImage.aspx?path="+path+"&src=camembert.gif'>
```

Cette page n'aura pas de partie graphique : l'intégralité du contenu produit sera géré par l'objet `Response`.

Fichier LoadImage.aspx (Version C#)

```
void Page_Load(Object sender,EventArgs e)
{
    string fileName = Request.QueryString["src"];
    string filePath = Request.QueryString["path"];
    string fullPath = filePath + "\\" + fileName;

    FileStream fileStream;
    long fileSize;
    fileStream = new FileStream(fullPath, FileMode.Open);
    fileSize = fileStream.Length;
    byte[] buffer = new byte[fileSize];
    fileStream.Read(buffer, 0, (int)fileSize);
    fileStream.Close();

    Response.BinaryWrite(buffer);
}
```

Récupération du nom du fichier image (src) et du chemin de stockage (path) afin de reconstituer le chemin complet du fichier (`fullPath`)

Chargement de l'image dans un objet de type `FileStream`

Écriture de l'image dans le flux de sortie

La version VB.NET de ce code est disponible en ligne à l'adresse :

- http://www.savonsdusoleil.com/src/vb/LoadImage.aspx

Pour pouvoir tester notre contrôle, il ne reste plus qu'à l'intégrer dans notre page d'analyse des ventes, ce que nous allons faire dans la section suivante.

Intégration du contrôle Camembert dans la page d'analyse des ventes

L'intégration de notre contrôle `Camembert` aux côtés du contrôle `VisualMap`, déjà présent dans la page, se déroulera en trois étapes :

- insertion d'une balise faisant référence au contrôle dans le contenu HTML, et ajout de la directive `Register` correspondante en haut de la page ;
- initialisation du contrôle dans le gestionnaire `Page_Load` ;
- modification de la fonction `ParametrerGraphique`, afin qu'elle gère l'affichage sélectif des contrôles `VisualMap` ou `Camembert` en fonction de l'axe d'analyse choisi par l'utilisateur.

La marche à suivre est la suivante :

1. Insérez une balise de contrôle serveur faisant référence au contrôle `Camembert`, dans l'onglet HTML de la page `Ventes.aspx`, à côté de la référence au contrôle `VisualMap` :

```
<SDS:Camembert id="Camembert" runat="server"/>
```

2. Ajoutez en haut de la page (onglet All), une directive `Register` permettant de faire référence au contrôle :

```
<%@ Register TagPrefix="SDS" Namespace="SDS.Controls"
    Assembly="SDS.Controls" %>
```

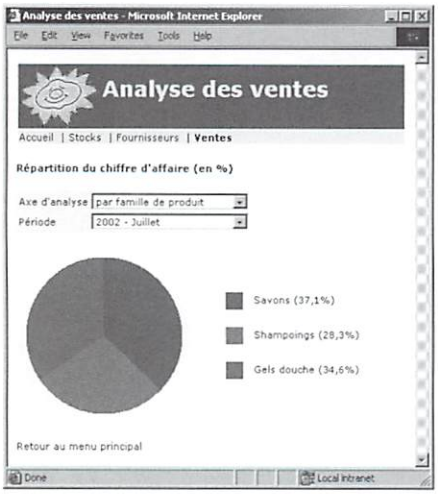


Figure 7–13 Page d'analyse des ventes par produit

3 Modifiez le gestionnaire d'événement `Page_Load` existant afin qu'il réalise l'initialisation du contrôle.

Fichier Ventes.aspx (Gestionnaire Page_Load / Version C#)

```
void Page_Load(Object sender, EventArgs e)
{
   if(!IsPostBack)
   {
      ...
      Camembert.DataValueField = "Pourcentage";
      Camembert.DataTextField = "NomFamille";
      ParametrerGraphique();
   }
}
```

On spécifie les noms des champs de la source de données à lier au contrôle

Enfin, ajoutez une fonction récupérant la liste des ventes mensuelles par famille pour un mois donné et modifiez la fonction `ParametrerGraphique` afin qu'elle gère le paramétrage sélectif des contrôles serveur spécifiques de la page.

```
DataView GetDataSource_Familles(int mois)
{
    string param = String.Format("2002{0:D2}",mois);
    string sql = " VentesMensuellesParFamille";
    SqlConnection myConnection = (SqlConnection)Session["myConnection"];
    SqlDataAdapter myAdapter = new SqlDataAdapter(sql,myConnection);
    myAdapter.SelectCommand.CommandType = CommandType.StoredProcedure;
    myAdapter.SelectCommand.Parameters.Add("@Mois",SqlDbType.VarChar);
    myAdapter.SelectCommand.Parameters["@Mois"].Value = param;
    DataTable myDataTable = new DataTable();
    myAdapter.Fill(myDataTable);
    return myDataTable.DefaultView;
}
void ParametrerGraphique()
{
  if (ListeAxes.SelectedIndex == 0)
  {
     Carte.DataSource = GetDataSource_Regions(ListeMois.SelectedIndex + 1);
     Camembert.Visible = false;
     Carte.Visible = true;
  }
  else
  {
     Camembert.DataSource = GetDataSource_Familles(ListeMois.SelectedIndex+1);
     Camembert.Visible = true;
     Carte.Visible = false;
  }
}
```

Cette fonction récupère une source de données de type `DataView` correspondant à l'exécution de la procédure `VentesMensuelles ParFamille` à laquelle on passe en paramètre le numéro du mois (par exemple : « 200207 »)

Si l'axe d'analyse « par région » est sélectionné, on paramètre le contrôle `Carte` et on masque le contrôle `Camembert`

Si l'axe d'analyse « par famille de produit » est sélectionné, on paramètre le contrôle `Camembert` et on masque le contrôle `Carte`

Pour finir, testez le résultat de l'exécution de la page, illustré figure 7-13 (n'oubliez pas de compiler le contrôle serveur spécifique et de créer un répertoire `img` sous la racine de l'application, si ce n'est pas déjà fait).

En résumé...

Dans ce chapitre, nous avons présenté le mécanisme des contrôles serveur spécifiques, qui permettent d'étendre les possibilités de la bibliothèque standard fournie avec ASP.NET :

- nous avons utilisé un contrôle serveur spécifique du marché (`VisualMap`) pour réprésenter les ventes par région, en soulignant au passage l'intégration de contrôles spécifiques à l'environnement de développement Web Matrix ;
- puis, nous avons développé notre propre contrôle serveur spécifique (graphique divisé en secteurs, de type camembert), en utilisant la classe de base `WebControl` et les possibilités de l'espace de nommage `System.Drawing` en matière de génération dynamique d'image.

Dans le chapitre suivant, nous allons illustrer les possibilités d'ASP.NET en matière de services Web, en implémentant un service de mise à jour des stocks, puis en ajoutant à notre intranet un module météo, qui récupérera des informations en temps réel auprès d'un service externe.

Exposer et utiliser des services Web

8

ASP.NET

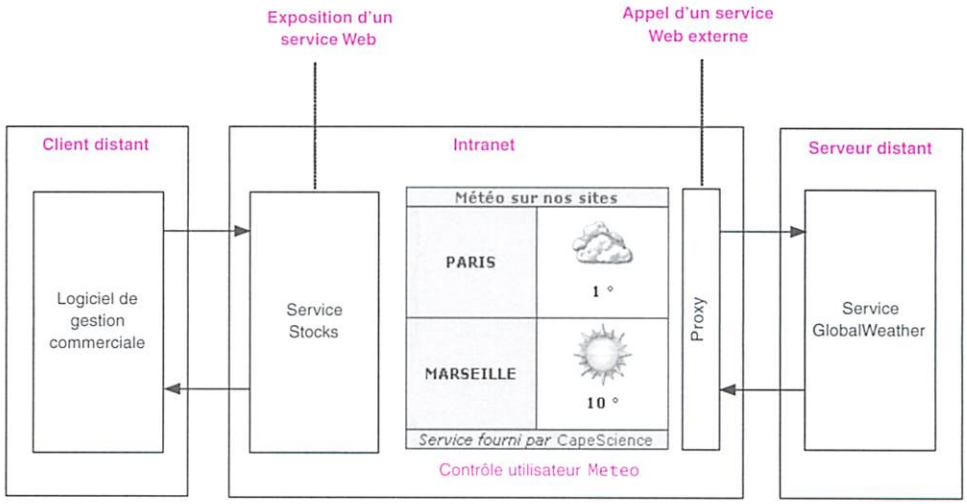
Service Web | Fichier .asmx | WSDL | UDDI | SOAP | Proxy

SOMMAIRE

- Implémentation d'un service Web
- Utilisation d'un service Web externe

MOTS-CLÉS

- Service Web
- Fichier .asmx
- WSDL
- UDDI
- SOAP
- Proxy



Dans ce chapitre, nous faisons un tour d'horizon des possibilités offertes par ASP.NET en matière de services Web, à travers l'implémentation d'un service Web de mise à jour des stocks et le développement d'un contrôle utilisateur affichant la météo actuelle pour les différents sites de l'entreprise (Paris et Marseille) à partir d'informations fournies par un service Web externe.

Implémentation d'un service Web de mise à jour des stocks

L'étude de cas que nous avons développée jusqu'à présent travaille sur un jeu de données fixe (produits en stock et résultats des ventes). Dans un cas réel, notre système de suivi des stocks et d'analyse des ventes ne présenterait que peu d'intérêt s'il n'était pas mis à jour régulièrement à partir du logiciel de gestion commerciale de l'entreprise.

Il y a quelques années, la mise en place d'échanges de données entre deux applications était généralement fastidieuse : un échange par fichier nécessitait le développement ou le paramétrage spécifique de modules d'import/export et présentait de nombreux inconvénients (données non structurées, absence d'échanges en temps réel, nécessité d'un administrateur pour traiter les cas d'erreur) ; une intégration spécifique nécessitait la modification des applications concernées, avec tous les désagréments et risques associés (dépendance forte entre les applications nuisant à la modularité du système d'information, emploi de protocoles spécifiques pas nécessairement acceptés par les logiciels pare-feu) ; enfin, le recours à un logiciel de type EAI (non représenté figure 8-1) représentait un investissement important.

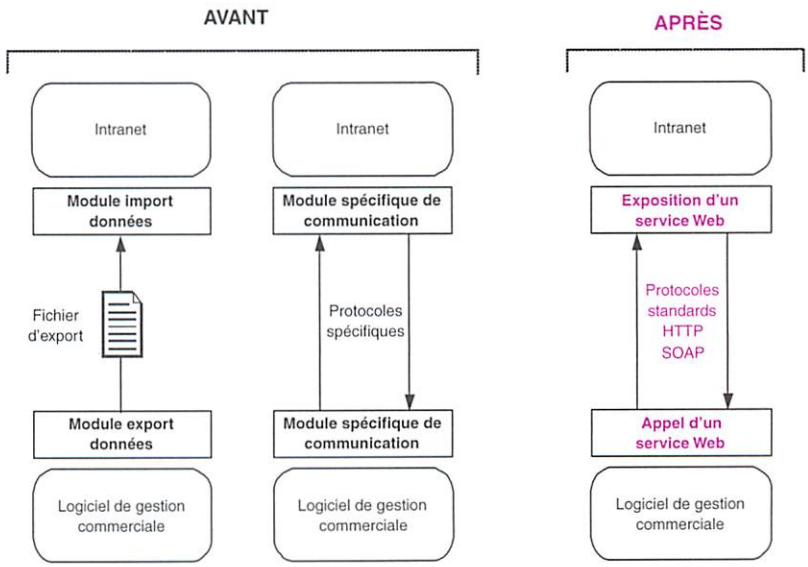


Figure 8–1 Quelques scénarios d'échanges de données entre applications

Désormais, l'utilisation de services Web constitue la meilleure solution pour mettre en œuvre des échanges de ce type : grâce à l'adoption de mécanismes et protocoles standards (SOAP/HTTP pour le transport des messages, WSDL pour la description des services, UDDI pour leur référencement dans des

Plus d'informations sur les services Web

Ce chapitre n'a pas pour vocation de traiter un sujet aussi vaste et complexe que les services Web mais uniquement d'illustrer leur mise en œuvre dans le cadre de la technologie ASP.NET. Pour plus d'informations, nous conseillons au lecteur de se référer à l'ouvrage suivant :

📖 *Services Web avec SOAP, WSDL, UDDI,* J.-M. Chauvet, Eyrolles 2002

annuaires), ils permettent à des applications de communiquer entre elles en s'affranchissant des contraintes d'intégration imposées par les solutions citées plus haut.

Ainsi, pour rendre possible la mise à jour de la base de données de l'intranet de notre entreprise fictive depuis un système externe, nous allons implémenter un service Web qui permettra la prise en compte de mouvements de stocks (nombre de produits vendus ou approvisionnés sur une période donnée).

Création d'un service Web avec ASP.NET

Un service Web mettant une ou plusieurs méthodes à la disposition de l'utilisateur, il est naturel de l'implémenter sous la forme d'une classe. Pour être exposée en tant que service, cette classe doit être située au sein d'un fichier portant l'extension `.asmx` et comportant une directive `WebService` ; d'autre part, les méthodes exposées doivent être précédées de l'attribut `WebMethod`.

Dans notre cas, nous souhaitons exposer une méthode `AjouterMouvementStock` qui permette à un système externe de transmettre des informations sur les mouvements de stock d'un produit donné sur une période donnée ; elle devra donc prendre en paramètre :

- le numéro du produit concerné ;
- le mois et l'année concernés ;
- la quantité de produit (positive pour un approvisionnement, négative pour une vente).

Cette méthode renverra une chaîne de caractères indiquant le résultat de l'opération (« Opération réussie » ou un message spécifiant le détail de l'éventuelle erreur survenue).

Cette méthode va être encapsulée au sein d'une classe `ServiceStocks` représentant le service Web, elle-même située au sein d'un fichier portant l'extension `.asmx`, comme le montre l'exemple de code qui suit.

Fichier ServiceStocks.asmx (Version VB.NET)

```
<%@ WebService language="VB" class="ServiceStocks" %>

Imports System
Imports System.Web.Services

Public class ServiceStocks

    <WebMethod> _
    Public Function AjouterMouvementStock( produitID As Int32, _
                                           annee As Int32 _
                                           mois As Int32, _
                                          quantite As Int32) As String

  ' Code de la function
    End Function
End Class
```

Cette directive est indispensable pour spécifier le langage utilisé dans la page et le nom de la classe implémentant le service.

Il est nécessaire d'importer l'espace de nommage `System.Web.Services`, notamment pour permettre l'utilisation de l'attribut `<WebMethod>`.

L'attribut `<WebMethod>` indique que la fonction est exposée en tant que méthode publique du service (noter le caractère _ indispensable, l'attribut devant être situé sur la même ligne que la déclaration de la fonction).

À propos des types utilisables dans un service Web

Les services Web permettent l'utilisation de très nombreux types de paramètres : types simples (entier, chaîne de caractères, booléen, etc.), tableaux de types simples ou, plus généralement, toute classe ou structure définie par l'utilisateur, dans la mesure où elle peut être représentée dans un schéma XSD.

DANS UN CAS RÉEL Mise à jour quotidienne

Dans notre étude de cas, nous avons choisi d'effectuer des remontées mensuelles d'informations de stocks pour des raisons de simplification. Dans un cas réel, ces informations seraient plutôt transmises quotidiennement ou hebdomadairement.

La version correspondante en C# n'est pas très différente (l'attribut `[WebMethod]` est, cette fois, encadré par des crochets et peut être situé sur la ligne précédant la déclaration de fonction).

Fichier ServiceStocks.asmx (Version C#)

```
<%@ WebService language="C#" class=" ServiceStocks " %>

using System;
using System.Web.Services;

public class ServiceStocks {
    [WebMethod]
    public string AjouterMouvementStock(   int produitID,
                                           int annee,
                                           int mois,
                                           int quantite)
    {
     // Code de la fonction
    }
}
```

Passons maintenant à l'implémentation de la fonction `AjouterMouvementStock`, qui va tout simplement insérer une nouvelle ligne dans la table `MouvementStock` de la base.

Utilisation de la classe de base WebService pour avoir accès à l'objet Session

Si l'implémentation de l'unique méthode de notre service ne présentera pas de difficulté particulière (utilisation d'un objet `DataTable` et d'un objet `SqlDataAdapter` pour réaliser l'insertion d'une nouvelle ligne dans une table), elle soulève néanmoins un problème : est-il possible d'accéder aux objets globaux de l'application ASP.NET, notamment l'objet `Session` qui maintient en cache la connexion vers la base de données ?

Dans l'état actuel de notre classe `ServiceStocks`, la réponse est non. Heureusement, une légère modification va nous permettre de contourner ce problème ; pour rendre l'objet global `Session` disponible au sein d'un service Web, il suffit de :

- faire dériver la classe du service de la classe de base `WebService` ;
- positionner à `True` la propriété `EnableSession` dans les attributs `WebMethod` correspondant aux méthodes au sein desquelles on souhaite accéder à l'objet `Session`.

WebService est une classe de base optionnelle

Notons qu'il n'est pas obligatoire de recourir à `WebService` comme classe de base ; celle-ci n'est indispensable que si l'on souhaite accéder aux objets globaux ASP.NET (`Session`, `Application`, etc.) au sein du service.

Fichier ServiceStocks.asmx (Version VB.NET)

```
<%@ WebService language="VB" class="ServiceStocks" %>

Imports System
```

```
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web.Services

Public class ServiceStocks : Inherits WebService



    <WebMethod(EnableSession:=True)> _
    Public Function AjouterMouvementStock(  produitID As Int32, _
                                            annee As Int32 _
                                            mois As Int32, _
                                          quantite As Int32) As String
      Try

        Dim myConnection As SqlConnection
        Dim myAdapter As SqlDataAdapter
        Dim myBuilder As SqlCommandBuilder
        Dim myDataTable  As DataTable
        Dim sql As String

        myConnection = CType(Session("myConnection"),SqlConnection)

        sql  = "SELECT * FROM MouvementStock WHERE ID_MouvementStock = 0"
        myDataTable = new DataTable()
        myAdapter = new SqlDataAdapter(sql,myConnection)
        myAdapter.Fill(myDataTable)

        myBuilder = new SqlCommandBuilder(myAdapter)

        Dim dateMouvement As string  = String.Format("{0:D2}/{1}",mois,annee)
        Dim myDataRow As DataRow  = myDataTable.NewRow()

        myDataRow("ID_Produit")     = produitID
        myDataRow("DateMouvement") = dateMouvement
        myDataRow("Quantite")       = quantite

        myDataTable.Rows.Add(myDataRow)

        myAdapter.Update(myDataTable)

      Catch e As Exception
        Return e.Message
      End Try

      Return "Opération réussie"

    End Function
End Class
```

Ces deux espaces de nommage supplémentaires sont importés pour permettre de référencer les classes d'accès aux données.

On spécifie que la classe `ServiceStocks` dérive la classe `WebService`, qui contient, entre autres, une propriété `Session`.

Le paramètre `EnableSession` est obligatoire pour que la propriété `Session` de la classe de base soit correctement initialisée.

Grâce à la classe de base et à `EnableSession`, on peut maintenant accéder à l'objet `Session`.

On insère une nouvelle ligne dans la table `MouvementStock` en utilisant le mécanisme déjà détaillé au chapitre 5 (allocation d'un objet `DataTable` auquel on ajoute une ligne, puis synchronisation avec la base de données grâce à un objet `SqlDataAdapter`, lui-même initialisé grâce à `SqlCommandBuilder`). NB : dans un cas réel, on vérifierait que les paramètres d'entrée sont valides (mois, année, numéro de produit) et que les informations de stocks ne sont pas saisies deux fois pour le même produit et la même période.

Un gestionnaire d'exception permet d'intercepter les éventuelles erreurs.

La version C# de ce code est disponible en ligne à l'adresse :

- www.savonsdusoleil.com/src/cs/ServiceStocks.asmx

L'implémentation de notre service est terminée : nous allons maintenant tester son fonctionnement.

Test du service Web de mise à jour des stocks

Dans un cas réel, notre service Web de mise à jour des stocks serait probablement appelé depuis un système externe doté d'une architecture technique qui pourrait être complètement différente de celle de notre intranet basé sur ASP.NET. En effet, la seule contrainte technique pour le système appelant est de gérer les protocoles standards SOAP et HTTP, et cela devient progressivement le cas pour la majorité des infrastructures techniques.

Néanmoins, nous allons tester le fonctionnement de notre service Web de manière locale, en utilisant successivement deux techniques, dans le but d'illustrer les possibilités d'ASP.NET en matière d'appel de services Web :

- test à partir de la page associée par défaut au service, fournie par ASP.NET ;
- développement d'une classe proxy.

La page de test par défaut associée au service

Le moyen le plus rapide pour tester le fonctionnement d'un service développé avec ASP.NET est d'utiliser la page de test par défaut, accessible en saisissant directement l'URL de la page qui implémente le service :

▸ http://localhost/<VotreApplication>/ServiceStocks.asmx

Cette page présente la liste des opérations (méthodes) exposées par le service (figure 8-2) et propose un lien vers la description du service au format WSDL (figure 8-3), dont nous aurons l'occasion de parler dans la deuxième partie de ce chapitre.

En cliquant sur le lien AjouterMouvementStock, on est conduit vers la page de test de l'opération, qui permet d'invoquer la méthode `AjouterMouvementStock` en lui transmettant des paramètres saisis par l'utilisateur. Cette page, générée à partir du fichier `DefaultWsdlHelpGenerator.aspx` situé dans le répertoire `CONFIG`

WSDL

WDSL (Web Services Description Language) est un format de description des services Web basé sur XML. À chaque service doit être associé un fichier WSDL décrivant les opérations offertes par le service avec leurs paramètres d'entrée/sortie. Ce fichier est utilisé par les clients faisant appel au service.

Figure 8–2 Page d'accueil par défaut du service

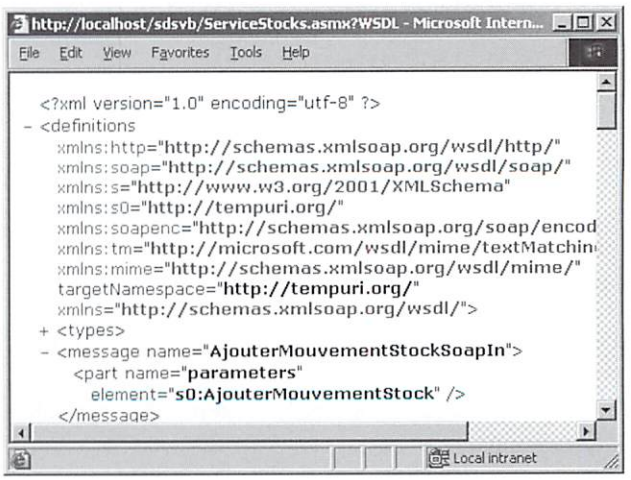


Figure 8–3 Description du service au format WSDL

du dossier d'installation du framework .NET, permet de tester l'appel du service depuis un navigateur ; elle utilise le protocole HTTP-GET et reçoit le résultat sous la forme d'un fichier XML (figure 8-4).

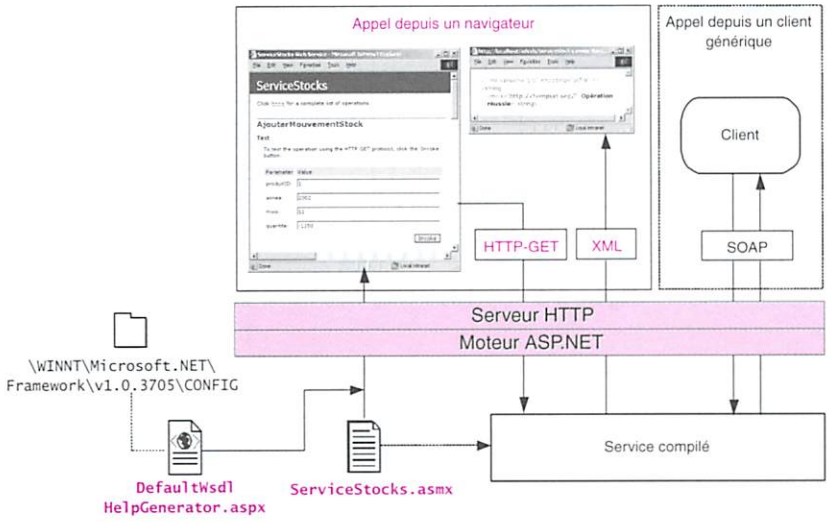


Figure 8–4 Appel du service depuis différents types de clients

Si cette technique constitue le moyen le plus rapide pour tester le bon fonctionnement d'un service Web développé avec ASP.NET, elle présente néanmoins un certain nombre d'inconvénients :

- elle ne peut pas être utilisée pour tester des services Web qui n'ont pas été développés avec ASP.NET ;
- elle ne fait pas appel au protocole standard SOAP, mais au protocole HTTP-GET ;
- elle ne permet pas de tester les options avancées comme la gestion de la sécurité à l'aide des en-têtes SOAP.

Une solution plus universelle pour réaliser l'appel d'un service Web consiste à développer un proxy, c'est-à-dire un objet local effectuant le lien avec le service : c'est ce que nous allons faire dans la section suivante.

Développement d'un proxy pour accéder au service Web

La partie fastidieuse dans la réalisation de l'appel d'un service Web est la préparation et le décodage des messages SOAP, protocole imposé pour la communication avec le service : heureusement, ASP.NET met à la disposition du développeur la classe `SoapHttpClientProtocol`, définie dans l'espace de nommage `System.Web.Services.Protocols`, qui permet d'encapsuler tout le travail pénible de génération et d'interprétation des messages SOAP au sein d'un proxy (voir figure 8-5).

Protocoles d'appel d'un service Web avec ASP.NET

Le protocole standard utilisé pour l'appel d'un service Web est SOAP. Néanmoins, les services Web développés par ASP.NET gèrent également, à des fins utilitaires, les protocoles HTTP-GET et HTTP-POST pour l'accès au service.

À propos des espaces de nommage

Chaque service Web public doit être associé à un espace de nommage, qui sert d'identifiant global unique (dit URI, pour Uniform Resource Identifier), afin d'éviter les collisions de noms. Si l'URI d'un service Web est généralement de la forme http:// adresse, cela ne signifie pas nécessairement qu'un mécanisme quelconque soit physiquement implémenté à cette adresse : le but est uniquement de disposer d'une chaîne unique. Par défaut, l'espace de nommage http://tempuri.org (pour temporary URI) est associé au service ; pour spécifier une valeur différente, il faut utiliser l'attribut `<WebService>` décrit dans la documentation ASP.NET.

SOAP

SOAP (Simple Object Access Protocol) est un protocole de communication entre applications basé sur XML : un message SOAP est constitué d'une enveloppe contenant un en-tête (facultatif) et le corps du message, le tout étant encapsulé dans une requête ou une réponse HTTP.

Proxy

Un proxy (qui signifie en anglais : délégué, mandataire) est un objet situé sur la machine appelante qui représente le service situé sur la machine distante : il offre la même interface que le service, avec lequel il gère de manière interne la communication (sous forme de messages SOAP, pour un service Web). Ce terme est probablement familier aux habitués des architectures distribuées (CORBA, DCOM, etc.).

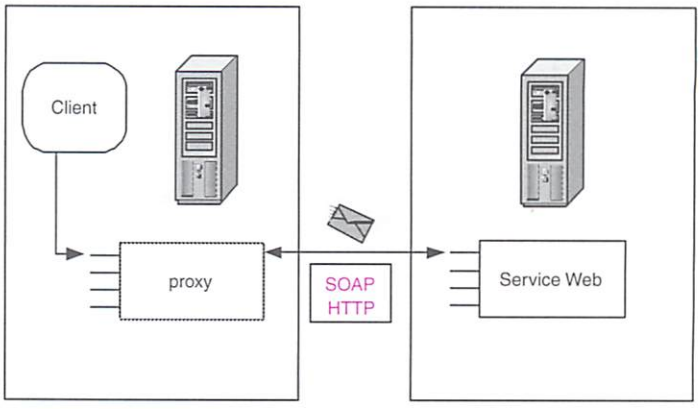


Figure 8–5 Appel du service Web via un proxy

La classe `SoapHttpClientProtocol` sert de classe de base pour l'implémentation du proxy, avec les règles suivantes :

- le constructeur de la classe dérivée doit spécifier l'URL du service auquel le proxy est associé, à l'aide de la propriété `URL` ;
- une méthode doit être associée à chaque opération du service, qu'elle appellera à l'aide de la méthode `Invoke` ;
- la classe proxy doit être précédée de l'attribut `<WebServiceBindingAttribute>` ;
- chaque méthode doit être précédée de l'attribut `<SoapDocumentMethodAttribute>`.

On propose ici un exemple de proxy associé à notre service de mise à jour des stocks.

Fichier ServiceStocksProxy.vb (Version VB.NET)

```
Imports System
Imports System.Web.Services
Imports System.Web.Services.Protocols

Namespace SDS
  <WebServiceBindingAttribute(Name:="ServiceStocks", _
    Namespace:="http://localhost/ServiceStocks.asmx?WSDL")> _
    Public Class ServiceStocksProxy : Inherits SoapHttpClientProtocol

    Public Sub New()
      MyBase.New
      Me.Url = "http://localhost/ServiceStocks.asmx"
    End Sub
```

La classe, qui dérive de `SoapHttpClientProtocol`, doit être précédée de l'attribut `WebServiceBindingAttribute` au sein duquel on précise le nom et l'espace de nommage associés aux informations de liaison du service (en l'occurrence, le document WSDL associé).

Il est indispensable d'effectuer le lien vers le service en spécifiant son point d'accès à l'aide de la propriété `Url` (NB : on fait ici l'hypothèse que le service est installé sous la racine du serveur Web local. Un véritable proxy aura plutôt vocation à être associé à un service distant !).

```
    <SoapDocumentMethodAttribute(_
      "http://tempuri.org/AjouterMouvementStock",_
      RequestNamespace:="http://tempuri.org/",  _
      ResponseNamespace:="http://tempuri.org/")>  _
      Public Function AjouterMouvementStock(ByVal produitID As Int32, _
      ByVal annee As Int32, _
      ByVal mois As Int32, _
      ByVal quantite As Int32) As String

      Dim params() As Object = { produitID, annee, mois, quantite }
      Dim results As Object =  Me.Invoke("AjouterMouvementStock",params)
      Return CType(results(0),String)
    End Function
  End Class

End Namespace
```

Implémentation de la méthode correspondant à l'unique opération proposée par le service : elle doit être précédée de l'attribut `SoapDocumentMethodAttribute`, pour lequel on spécifie l'URI du service ainsi que les paramètres `RequestNamespace` et `ResponseNamespace`. Cette méthode stocke les paramètres d'entrée (passés par valeur pour une meilleure performance) dans un tableau d'objets qui est ensuite transmis à la fonction `Invoke` de la classe de base, qui réalise l'appel du service. Les paramètres de sortie sont également reçus sous la forme d'un tableau d'objets.

La version C# de ce code est disponible en ligne à l'adresse :

- www.savonsdusoleil.com/src/cs/ServiceStocksProxy.cs

Pour pouvoir utiliser ce proxy, il faut le compiler, puis y faire référence depuis une page ASP.NET permettant la saisie des paramètres d'entrée et l'affichage du résultat : nous allons avoir l'occasion de détailler ce processus dans la seconde partie de ce chapitre, où nous utiliserons un proxy pour faire appeler un service Web distant fournissant des informations météorologiques.

Génération automatique d'un proxy avec WSDL

L'implémentation manuelle d'un proxy peut vite devenir fastidieuse, notamment dans le cas d'un service exposant de nombreuses opérations ayant chacune de nombreux paramètres. Heureusement, l'utilitaire `wsdl` fourni avec le framework .NET permet de générer automatiquement le code d'un proxy à partir d'un fichier WSDL, comme nous allons le montrer dans la suite de ce chapitre.

Affichage de la météo en faisant appel à un service Web externe

Dans la première partie de chapitre, nous avons vu comment exposer un service Web avec ASP.NET ; passons maintenant de l'autre côté de la barrière en implémentant un contrôle utilisateur qui affichera la météo à Paris et à Marseille (les implantations de notre entreprise fictive) à partir d'informations fournies par un service Web externe. Nous allons réaliser toutes les opérations nécessaires à l'appel du service : depuis sa recherche sur Internet à l'aide d'un annuaire UDDI jusqu'à sa mise en œuvre en utilisant un proxy.

Recherche d'un service fournissant des informations météorologiques

Les services Web disponibles sur Internet sont regroupés dans des annuaires mondiaux dotés de moteurs de recherche (annuaires UDDI), accessibles à l'adresse www.uddi.org

UDDI

UDDI (Universal Description, Discovery and Integration) est un standard d'annuaires de services Web. Actuellement, quatre implémentations sont disponibles :

- l'annuaire de Microsoft (uddi.microsoft.com),
- celui d'IBM (uddi.ibm.com),
- celui de SAP (uddi.sap.com),
- et enfin celui de NTT (www.ntt.com/uddi).

Dans notre cas, nous recherchons un service capable de fournir des informations météorologiques (en anglais : *weather*) ; nous allons donc effectuer une recherche de la manière suivante :

- Rendez-vous sur un des annuaires UDDI disponibles (à partir de www.uddi.org).
- Saisissez le mot-clé « %Weather%» comme critère de recherche pour le nom du service (recherchera tous les services dont le nom contient « Weather », voir figure 8-6).

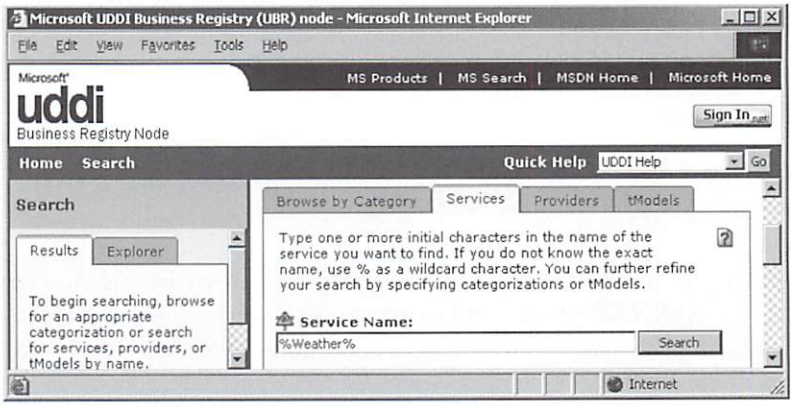


Figure 8–6 Recherche d'un service Web à l'aide d'un annuaire UDDI

Figure 8–7 Résultats de la recherche

Les résultats de la recherche dépendent bien évidemment de la pertinence des mots-clés choisis et des services disponibles, comme pour un moteur de recherche standard. En l'occurrence, nous avons de la chance car il semble exister un service *GlobalWeather* qui fournit des informations météorologiques au niveau mondial (voir figure 8-7).

La consultation des informations associées au service dans l'annuaire UDDI nous fournit deux données importantes : le point d'accès au service, qui indique où le service est physiquement accessible ainsi que la localisation du fichier WSDL correspondant au service, indispensable pour connaître l'interface d'appel du service.

Entité	Localisation
Point d'accès au service	http://live.capescience.com:80/ccx/GlobalWeather
Fichier WSDL correspondant	http://live.capescience.com/wsdl/GlobalWeather.wsdl

L'examen du fichier WSDL (voir figure 8-8) nous permet de « découvrir » – selon le terme consacré – les caractéristiques principales du service :

- la fonction principale est `getWeatherReport` : elle permet d'obtenir un rapport météorologique pour une localisation donnée (en pratique, un aéroport, pour lequel il faut saisir le code ICAO) ;

À propos des codes ICAO

Les codes ICAO (International Civil Aviation Organisation) permettent d'identifier tous les aéroports civils du globe (par exemple : LFML désigne l'aéroport de Marseille-Marignane et LFPO celui de Paris-Orly). Nous allons utiliser ces codes avec le service GlobalWeather pour spécifier les lieux pour lesquels on souhaite obtenir des informations. Pour plus d'informations, voir :

- www.icao.int

- le rapport contient de nombreuses informations ; nous n'en utiliserons que deux : la température (propriété `temperature`) et le taux de couverture (propriété `sky`).

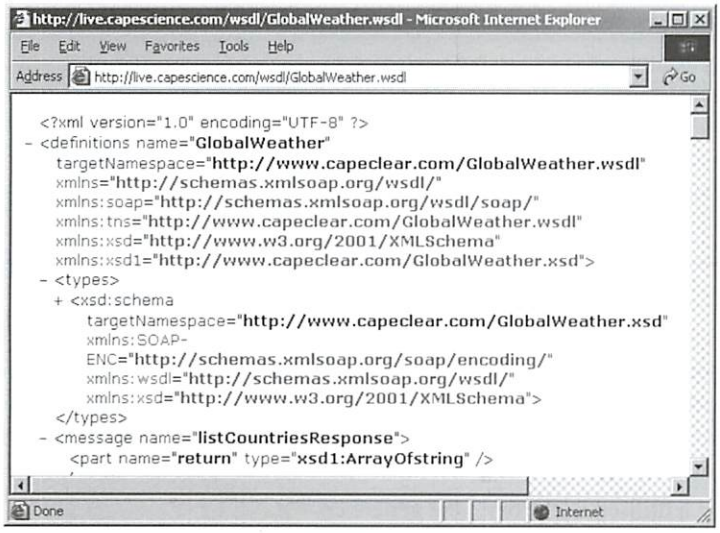


Figure 8–8 Fichier WSDL associé au service GlobalWeather

Nous avons désormais suffisamment d'information pour passer à l'étape suivante : la génération d'un proxy pour le service *GlobalWeather*.

Génération d'un proxy pour le service GlobalWeather

Nous avons déjà eu l'occasion de réaliser manuellement un proxy dans la première partie de ce chapitre : nous allons maintenant voir comment générer automatiquement un proxy pour le service *GlobalWeather* à partir d'un fichier WSDL et de l'utilitaire `WSDL` :

1. Ouvrez une fenêtre DOS.
2. Placez-vous dans le répertoire de votre application.
3. Exécutez l'utilitaire WSDL avec la syntaxe suivante (voir figure 8-9) :

```
wsdl [url] /l:[language] /n:[namespace]
```

où :

- `[url]` spécifie l'emplacement du document WSDL pour lequel on souhaite générer un proxy (en l'occurrence : http://live.capescience.com/wsdl/GlobalWeather.wsdl) ;
- `[language]` spécifie le langage dont lequel vous souhaitez générer le proxy (`cs` pour C#, `vb` pour VB.NET) ;
- `[namespace]` spécifie l'espace de nommage dans lequel sera contenu la classe générée ; par exemple : `WeatherServices`.

Variables d'environnement

Si vous n'avez pas choisi d'enregistrer les variables d'environnement lors de l'installation du .NET SDK, ajoutez le chemin

`\Program Files\Microsoft.NET\FrameworkSDK\Bin`

dans la variable d'environnement `Path` de votre système, afin de pouvoir faire référence à l'utilitaire WSDL depuis n'importe quel répertoire.

Un fichier portant le nom du service et contenant une classe proxy dérivée de `SoapHttpClientProtocol` est alors automatiquement généré (l'ordinateur doit être connecté à Internet car l'utilisateur va récupérer les informations WSDL en ligne).

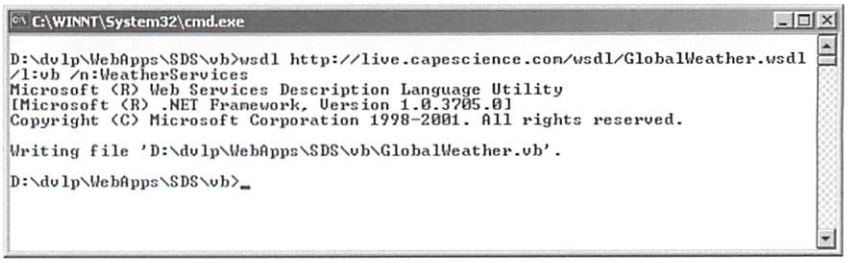


Figure 8–9 Génération d'un proxy avec WSDL

Pour pouvoir utiliser le proxy, il faut enfin le compiler sous la forme d'un assemblage privé `GlobalWeather.dll` (placé dans le répertoire `bin` de l'application), en utilisant le compilateur `csc` (pour C#) ou `vbc` (pour VB.NET) comme nous avons déjà eu l'occasion de le faire dans les chapitres précédents.

Syntaxe à utiliser si le proxy a été généré en VB.NET :

```
vbc /out:bin\GlobalWeather.dll /t:library GlobalWeather.vb
  ➥ /r:system.dll,system.web.services.dll,system.xml.dll
```

Syntaxe à utiliser si le proxy a été généré en C# :

```
csc /out:bin\GlobalWeather.dll /t:library GlobalWeather.cs
  ➥ /r:system.dll,system.web.services.dll,system.xml.dll
```

Pour finir, nous allons implémenter un contrôle utilisateur qui va afficher des informations météorologiques fournies par le service Web *GlobalWeather*, interrogé à l'aide du proxy que nous venons de réaliser.

Implémentation du contrôle utilisateur Meteo

RAPPEL Contrôle utilisateur

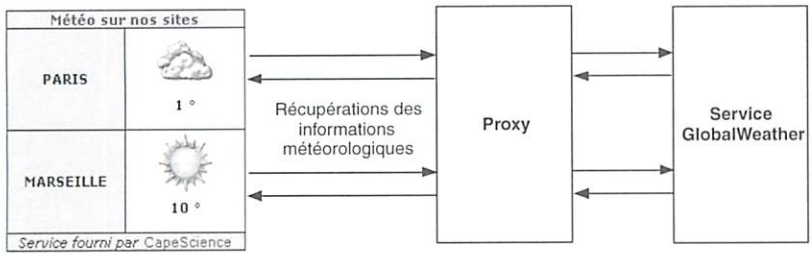
Un contrôle utilisateur est un morceau de page Web encapsulé dans un composant graphique réutilisable (voir chapitre 3).

Le contrôle utilisateur que nous nous proposons de réaliser affichera des informations météorologiques sommaires (température et taux de couverture du ciel) pour les deux sites de l'entreprise (Paris et Marseille), récupérées dynamiquement auprès du service GlobalWeather (voir figure 8-10).

Réalisation de la maquette du contrôle Meteo

Commençons par réaliser la maquette du contrôle utilisateur (voir figure 8-11), qui ne présente pas de difficultés particulières :

1. Démarrez Web Matrix.
2. Créez une nouveau contrôle utilisateur nommé `Meteo.ascx`.



Contrôle utilisateur `Meteo`

Figure 8–10 Architecture du contrôle utilisateur `Meteo`

3 Créez un tableau HTML comportant quatre lignes et deux colonnes (d'une largeur de cent pixels chacune) avec un couleur de fond jaune.

4 Fusionnez les deux cellules de la première ligne et saisissez le texte : « Météo sur nos sites ».

5 Dans la colonne de gauche de la deuxième ligne, saisissez le texte « PARIS » ; dans la colonne de droite, insérez deux contrôles serveur : un contrôle de type `Image` nommé `ImageParis` (pour l'état du ciel) et un contrôle de type `Label` nommé `TemperatureParis` (pour la température).

6 Répétez la même opération sur la troisième ligne en remplaçant Paris par Marseille.

7 Enfin, fusionnez les deux cellules de la dernière ligne et saisissez le texte « Service fourni par CapeScience », avec un lien vers l'URL www.capescience.com/webservices/globalweather (il est normal de citer le fournisseur de ce service gratuit !)

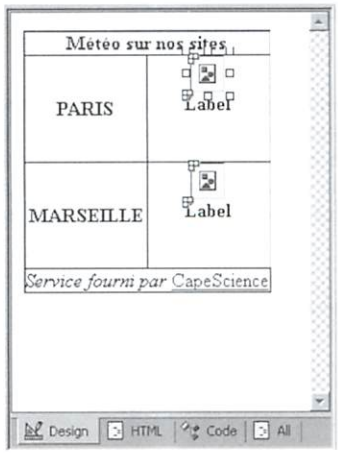


Figure 8–11 Maquette du contrôle utilisateur Meteo

Implémentation du contrôle Meteo

Grâce au proxy réalisé précédemment, l'appel du service Web va être facile à implémenter, puisqu'il va se résumer à un simple appel de fonction.

Avant tout, insérez une directive `Import` en haut du fichier afin de pouvoir faire référence au proxy (dont l'espace de nommage `WeatherServices` a été fixé lors de la génération automatique par WSDL) :

```
<%@ Import Namespace="WeatherServices" %>
```

Puis, implémentez le code de la page afin qu'il effectue les opérations suivantes lors du chargement de la page (événement `Page_Load`) :

- récupération des informations météo pour l'aéroport de Paris-Orly (code LFPO) ;
- récupération des informations météo pour l'aéroport de Marseille (code LFML) ;
- mise à jour des contrôles serveur correspondants.

Fichier Meteo.ascx (Version VB.NET)

Cette classe utilitaire permet de stocker les informations météorologiques relatives à un lieu (image et température).

```
Class WeatherInfos

    public imageUrl As String
    public temperature As Double

End Class
```

Cette fonction utilisateur récupère les informations météorologiques pour un lieu donné à partir du code ICAO.

```
Function GetWeatherInfos(code As String) As WeatherInfos
```

Création d'une instance du proxy (qui représente le service) et obtention du rapport météorologique pour la station demandée.

```
    Dim gw As GlobalWeather = new GlobalWeather()
    Dim report As WeatherReport = gw.getWeatherReport(code)

    Dim infos As WeatherInfos = new WeatherInfos()
    Dim tauxCouverture As Int32
```

Choix de l'image en fonction du taux de couverture (contenu dans `sky.layers(0).extend`). Les images, téléchargeables depuis le site de l'étude de cas, doivent être placées dans le répertoire `img`.

```
    If (report.sky.layers.Length>0) Then
        tauxCouverture = report.sky.layers(0).extent
        infos.imageUrl = String.Format("img/Couvert{0}-4.gif",
            ➥ tauxCouverture)
    Else
        infos.imageUrl = "img/AbsenceInfos.gif"
    End If
```

Récupération de la température.

```
    infos.temperature = report.temperature.ambient

    return infos

End Function
Sub Page_Load(sender As Object, e As EventArgs)

    Dim imageUrl As String
    Dim temperature As Double

    Dim infos As WeatherInfos
```

Récupération des informations météorologiques pour Paris et paramétrage des contrôles serveur correspondants.

```
    infos = GetWeatherInfos("LFPO")
    ImageParis.ImageUrl = infos.imageUrl
    TemperatureParis.Text = String.Format("{0} °",infos.temperature)
```

Récupération des informations météorologiques pour Marseille et paramétrage des contrôles serveur correspondants.

```
    infos = GetWeatherInfos("LFML")
    ImageMarseille.ImageUrl = infos.imageUrl
    TemperatureMarseille.Text = String.Format("{0} °",infos.temperature)
End Sub
```

La version C# de ce code est disponible en ligne à l'adresse :

- www.savonsdusoleil.com/src/cs/Meteo.ascx

Le résultat de l'exécution de ce contrôle, une fois intégré à la page d'accueil de l'intranet, est présenté figure 8-12.

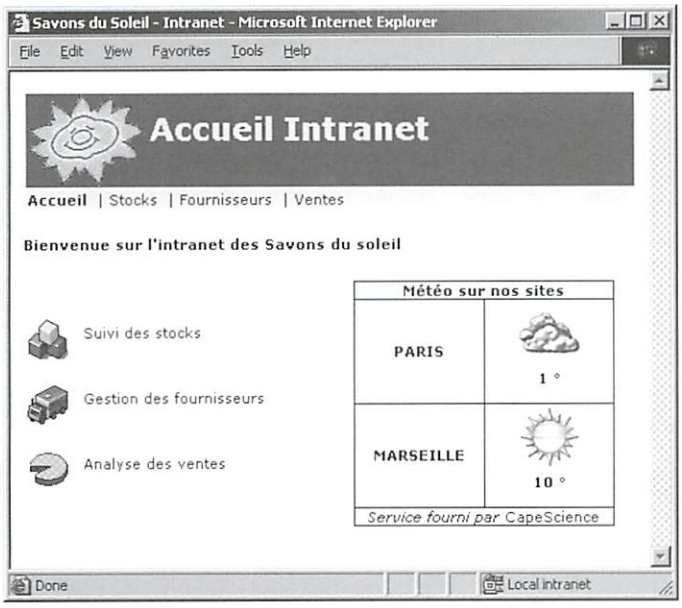


Figure 8–12 Page d'accueil avec intégration du contrôle Meteo

En résumé...

Dans ce chapitre, nous avons mis en œuvre quelques-unes des possibilités offertes par ASP.NET en matière de services Web :

- implémentation d'un service Web sous la forme d'un fichier `.asmx` ;
- utilisation de la classe de base `WebService` pour avoir accès aux objets intrinsèques ASP.NET (`Session`...) au sein du service ;
- test du service grâce à la page par défaut associée au service ;
- implémentation d'un proxy à l'aide de la classe de base `SoapHttpClientProtocol` ;
- génération automatique d'un proxy à l'aide de l'utilitaire WSDL ;
- implémentation d'un contrôle utilisateur faisant appel à un service Web externe.

Certaines fonctionnalités proposées par ASP.NET sur le sujet n'ont pas pu être abordées : citons, par exemple, le *screen-scraping*, la gestion des en-têtes SOAP, le maintien en cache des résultats d'appel d'un service ou encore la sécurisation des échanges.

Ce dernier thème sera abordé dans le prochain chapitre : sécurité, configuration, optimisation et déploiement des applications ASP.NET.

Sécurisation, optimisation et déploiement

9

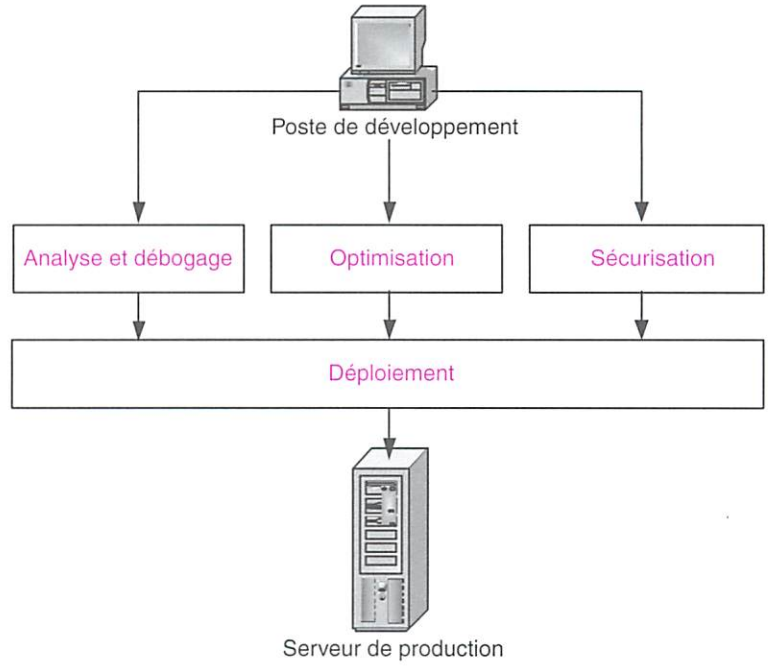
ASP.NET

Authentification | machine.config | web.config | en-tête SOAP | débogueur .NET | Application_Error |

SOMMAIRE

MOTS-CLÉS

- Authentification
- Autorisation
- machine.config
- web.config
- En-tête SOAP
- Trace
- Cache
- Débogueur .NET
- Application_Error
- Global Assembly Cache



Dans ce chapitre, nous examinons les sujets liés à la mise en production d'une application ASP.NET : authentification des utilisateurs et contrôle des autorisations, possibilités de configuration, analyse et débogage, gestion personnalisée des erreurs, optimisations des performances à l'aide des mécanismes de cache et déploiement vers un serveur de production.

Sécurisation d'une application ASP.NET

La sécurisation d'une application Web constitue un élément fondamental de l'architecture, notamment pour un intranet où il est nécessaire d'authentifier les utilisateurs, voire de leur attribuer des autorisations en fonction de leur profil.

La mise en place d'un mécanisme de contrôle d'accès nécessitait jusqu'alors l'implémentation de modules spécifiques : ASP.NET facilite le travail en fournissant un ensemble de fonctionnalités de sécurisation disponibles en standard.

Redirection automatique vers une page d'authentification

Une des possibilités d'ASP.NET en matière de sécurité est l'implémentation d'un mécanisme d'authentification automatique qui redirige les utilisateurs non authentifiés vers une page demandant la saisie d'un identifiant et d'un mot de passe (voir figure 9-1).

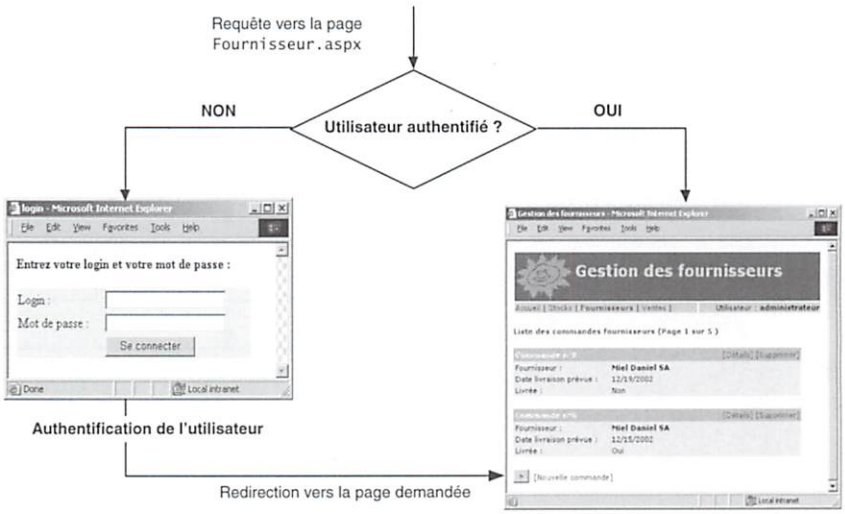


Figure 9–1 Authentification des utilisateurs

IMPORTANT Types de fichiers protégés

Seuls les éléments constitutifs d'une application ASP.NET (fichiers `.aspx`, `.ascx` et `.asmx`) sont concernés par ce mécanisme d'authentification, lequel ne contrôle pas, par exemple, l'accès à des images ou des fichiers texte situés dans le répertoire de l'application (ceci est imposé par la compatibilité ascendante avec ASP).

Pour mettre en œuvre ce mécanisme d'authentification, qui repose sur l'utilisation d'un *cookie* :

1. Réalisez une page nommée, par exemple, `login.aspx`, contenant deux zones d'éditions qui permettent respectivement la saisie d'un identifiant et d'un mot de passe et un bouton qui effectue l'établissement de la connexion, auquel on associera un gestionnaire d'événement décrit plus loin.
2. Modifiez le fichier `web.config` de l'application en lui rajoutant une section `<authentification>` qui spécifie le type d'authentification (`Forms` pour formulaire Web), la page utilisée pour l'authentification (`loginUrl`), la durée de vie du *cookie* associé à la session (en minutes) et la liste des identifiants et mots de passe autorisés à accéder à l'application.

```
<authentication mode="Forms">
  <forms name=".sds" loginUrl="login.aspx"
         protection="All" timeout="30" path="/">
    <credentials passwordFormat="Clear" >
       <user name="administrateur" password="password1"/>
       <user name="marketing" password password2"/>
       <user name="logistique" password=" password3"/>
    </credentials>
  </forms>
</authentication>
<authorization>
  <deny users="?"/>
</authorization>
```

La section `<credentials>` permet de spécifier la liste des utilisateurs autorisés à accéder à l'application. NB : les mots de passe peuvent également être cryptés ; voir à ce sujet la documentation de la méthode `HashPasswordForStoringInConfigFile`.

Cette section spécifie qu'il faut rejeter toutes les requêtes des utilisateurs non authentifiés.

3 Associez un gestionnaire à l'événement `Click` du bouton de connexion, qui réalise la validation des informations saisies grâce à la méthode `Authenticate` de la classe `FormsAuthentication` et redirige, en cas de succès, l'utilisateur vers la page demandée.

Ficher login.aspx (Version C#)

```
void OnClick_DoConnect(Object sender, EventArgs e)
{
    if (FormsAuthentication.Authenticate(UserID.Text,Password.Text))
    {
       FormsAuthentication.RedirectFromLoginPage(UserID.Text,false);
    }
    else
    {
       Message.Text = "Identification incorrecte. Merci de réessayer";
    }
}
```

La version VB.NET de ce code est disponible en ligne à l'adresse :

▸ http://www.savonsdusoleil.com/src/vb/login.aspx

Contrôle des autorisations en fonction de l'utilisateur

Une fois l'utilisateur authentifié, il est possible de gérer les autorisations accordées à l'utilisateur en fonction de son identifiant ou de son rôle, grâce aux informations fournies par l'objet `Context.User`, accessible au sein de l'application. On peut par exemple modifier le contrôle utilisateur `Barre de navigation` de manière à ce que :

- le nom de l'utilisateur soit affiché dans la barre de navigation ;
- l'utilisateur « administrateur » ait accès à l'ensemble des rubriques ;
- l'utilisateur « marketing » n'ait accès qu'aux rubriques Accueil et Ventes ;
- l'utilisateur « logistique » n'ait accès qu'aux rubriques Accueil, Stocks et Fournisseurs.

DANS UN CAS RÉEL

Sécurisation complète de l'infrastructure

Dans le cadre d'une application réelle, il faudrait aborder de manière plus complète la sécurisation de l'application : protection physique de l'accès au serveur, mise en place d'un pare-feu, authentification forte des utilisateurs, etc.

Pour cela :

1. Ajoutez le texte « Utilisateur : » suivi d'un contrôle serveur de type `Label` nommé `Utilisateur` en bas à droite du contrôle `Barre de navigation`.
2. Ajoutez le code suivant à la fin du gestionnaire `Page_Load` du fichier `NavBar.ascx` implémenté au chapitre 3 :

Les fichiers de configuration ASP.NET

La configuration des applications ASP.NET s'effectue par l'intermédiaire d'une hiérarchie de fichiers XML portant l'extension `.config` (figure 9-2) :

- Le fichier `machine.config`, placé dans le sous-répertoire CONFIG du dossier d'installation du framework .NET, spécifie les options de configurations appliquées par défaut à toutes les applications ASP.NET installées sur la machine ;
- Un fichier `web.config` (optionnel) placé sous le racine du serveur HTTP permet de spécifier des options de configuration particulières (prenant le pas sur le fichier `machine.config`) pour l'ensemble des applications gérées par ce serveur HTTP ;
- Des fichiers `web.config` placés dans les répertoires de chaque application permettent de spécifier des options de configuration particulières, qui prennent le pas sur les fichiers précédents.

Les mises à jour effectuées sur ces fichiers, lesquels présentent au passage l'avantage d'être faciles à lire et à éditer, sont prises en compte immédiatement sans nécessiter de redémarrage du serveur HTTP : les options de configuration sont conservées dans l'objet Cache d'ASP.NET, qui est automatiquement mis à jour lorsqu'une modification des fichiers de configuration est détectée ; un nouveau domaine d'application (*application domain*) est alors automatiquement instancié par le CLR pour traiter les requêtes des nouveaux utilisateurs en appliquant les nouvelles options, tandis que l'ancien domaine est conservé tant qu'il y a des utilisateurs qui lui sont connectés.

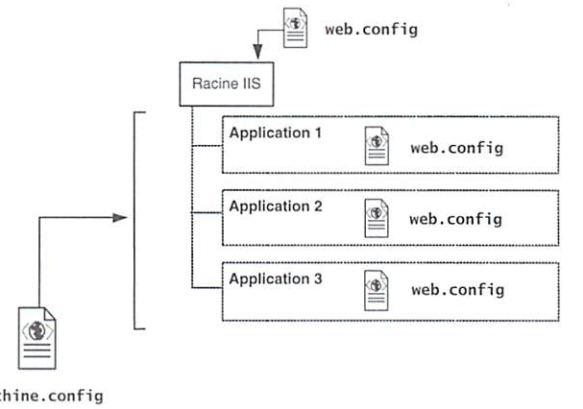


Figure 9–2 Fichiers de configuration ASP.NET

Le redémarrage d'IIS n'est plus nécessaire

Les applications ASP étant basées sur la métabase IIS, la prise en compte d'un changement de configuration nécessitait un redémarrage d'IIS : le mécanisme des fichiers de configuration ASP.NET supprime cette contrainte.

Format d'un fichier .config (simplifié)

L'élément de plus haut niveau doit être de type `<configuration>`.

La section `<system.web>` contient les principales options relatives à la configuration de l'application (sécurité, internationalisation, gestion des erreurs, etc.).

La section `<appSettings>` est généralement utilisée pour spécifier des paramètres relatifs à l'application que l'on souhaite pouvoir facilement modifier (chaîne de connexion à la base de données, nom du serveur SMTP, etc.).

```
< ?xml version= "1.0" encoding="UTF-8" ?>
<configuration>

  <system.web>
   ...
  </system.web>

  <appSettings>
     ...
  </appSettings>
</configuration>
```

Fichier NavBar.ascx (Gestionnaire Page_Load / Version C#)

```
...
string userName = Context.User.Identity.Name;
Utilisateur.Text = userName;

if (userName == "marketing")
{
   if ((SelectedIndex ==1) || (SelectedIndex == 2))
      Response.Redirect("login.aspx");
   Rubrique2.Visible = false;
   Rubrique3.Visible = false;
}
if (userName == "logistique")
{
   if ((SelectedIndex ==3))
      Response.Redirect("login.aspx");
   Rubrique4.Visible = false;
}
```

On récupère le nom de l'utilisateur et on l'affiche dans la barre de navigation.

Si l'utilisateur est « marketing », on masque les rubriques Stocks et Fournisseurs (on vérifie également la valeur de l'index de la rubrique, pour interdire l'accès à une page non autorisée pour laquelle l'utilisateur aurait saisi directement l'adresse dans le navigateur).

Si l'utilisateur est « logistique», on masque la rubrique Ventes.

La version VB.NET de ce code est disponible en ligne à l'adresse :

› http://www.savonsdusoleil.com/src/vb/NavBar.ascx

Le résultat de l'exécution pour l'utilisateur « marketing » est présenté figure 9-3.

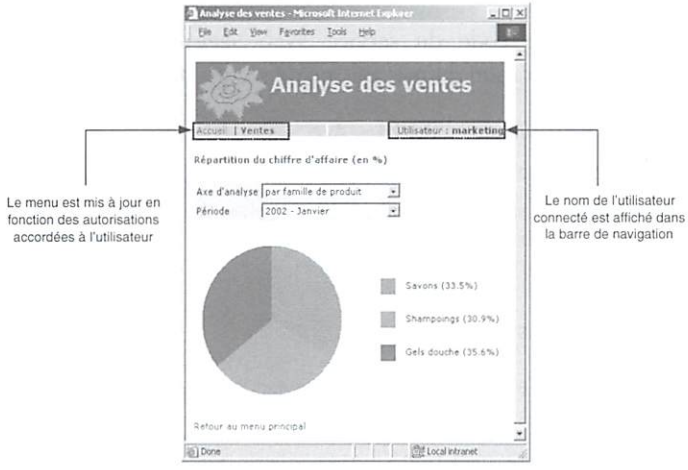


Figure 9–3 Gestion sélective des autorisations

Sécurisation d'un service Web

À l'image des applications Web s'exécutant dans un navigateur, les services Web nécessitent généralement un mécanisme de sécurisation : par exemple, il paraît indispensable de contrôler l'identité des programmes qui réalisent une mise à jour des stocks par l'intermédiaire du service développé dans le chapitre précédent.

Cette authentification peut être réalisée grâce à la transmission des informations relatives à l'utilisateur (identifiant et mot de passe) dans l'en-tête des messages SOAP, comme l'illustre la figure 9-4.

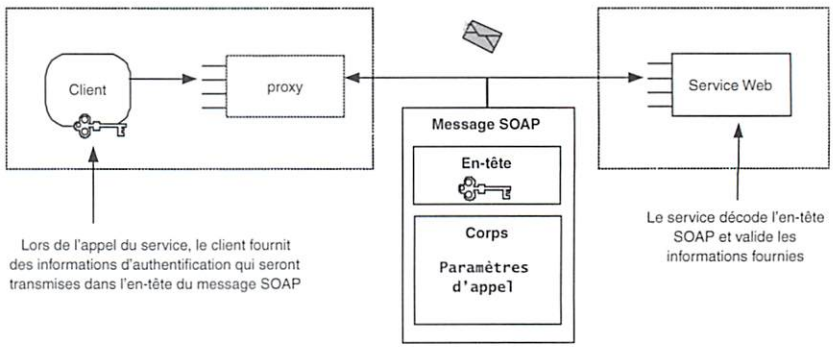


Figure 9–4 Sécurisation de l'appel d'un service Web

La sécurisation d'un service Web se déroulera en trois étapes :

- définition d'une classe `AuthHeader`, dérivant de `SoapHeader`, contenant les informations d'authentification (identifiant et mot de passe) ;
- spécification du fait qu'un en-tête SOAP de type `AuthHeader` est requis pour l'appel du service ;
- vérification des informations d'authentification au début de l'exécution du service.

Voici par exemple le code du service Web `ServiceStocks`, développé au chapitre 8, après intégration du mécanisme de sécurisation :

ServicesStocks.asmx (Version VB.NET – Avec sécurisation)

```
Public class ServiceStocks : Inherits WebService

   Public Class AuthHeader : Inherits SoapHeader
      Public Username As String
      Public Password As String
   End Class

   Public sHeader As AuthHeader

   <WebMethod(EnableSession:=True),SoapHeader("sHeader")>

   Public Function  AjouterMouvementStock(...)

       If (sHeader Is Nothing)
          Return "Erreur : absence d'informations d'authentification"
       End If

       If (sHeader.Username <> "<YourID>") Or
             ➥ (sHeader.Password <> "<YourPwd")
          Return "Erreur : authentification incorrecte"
        End If
        ...
    End Function
```

On déclare une classe `AuthHeader` dérivant de `SoapHeader`, destinée à contenir un identifiant et un mot de passe.

On déclare une variable membre de type `AuthHeader`, dont on spécifie qu'elle est requise dans l'en-tête SOAP du message.

On vérifie la présence d'informations d'authentification et la validité des informations transmises.

La version C# de ce code est disponible en ligne à l'adresse

▸ http://www.savonsdusoleil.com/src/cs/ServiceStock.asmx

Du côté client, il suffit ensuite de générer le proxy correspondant (grâce à l'utilitaire WSDL, voir chapitre précédent) et d'utiliser la propriété AuthHeaderValue (générée du fait de la présence de l'attribut SoapHeader) pour spécifier les informations d'authentification à transmettre au service :

```
Dim proxy As ServiceStocks = new ServiceStocks()
Dim authInfo As AuthHeader = new AuthHeader
authInfo.Username = "login"
authInfo.Password = "password"
proxy.AuthHeaderValue = authInfo
proxy.AjouterMouvementStock(...)
```

Débogage et gestion des erreurs

La gestion des erreurs constituant un élément très important de la qualité d'une application, ASP.NET met à la disposition du développeur un certain nombre de techniques qui permettent l'analyse, le débogage ainsi que le traitement spécifique des exceptions :

- le mécanisme de trace et le débogueur fourni avec le kit de développement .NET permettent d'analyser en détail l'exécution d'une application ASP.NET ;
- le mécanisme de gestion des exceptions permet de traiter de manière structurée l'ensemble des cas d'erreurs pouvant survenir dans une application ;
- les options de configuration disponibles rendent possible le traitement personnalisé des erreurs (redirection vers une page spécifique, envoi de messages à l'administrateur, etc.).

Analyser l'exécution d'une application

Deux techniques sont possibles pour tracer l'exécution d'une application : l'utilisation de l'option Trace, très facile à mettre en œuvre mais limitée à l'affichage de messages décrivant les différentes étapes de l'exécution d'une page, et le recours au débogueur .NET, qui permet d'atteindre un niveau d'analyse avancé.

Accéder rapidement au déroulement détaillé avec l'option Trace

L'option Trace de la directive Page est le moyen le plus simple d'avoir une vision globale de l'exécution d'une page : lorsqu'elle est activée, un résumé complet des éléments liés à la requête (heure, identifiant de session, contenu de l'objet Session, encodage de la réponse...) et à l'exécution de la page (heures de début et de fin de l'exécution des principaux événements prédéfinis) est ajouté à la suite du contenu HTML produit.

En outre, l'utilisateur a la possibilité de rajouter des messages de trace au sein du code grâce aux méthodes Write et Warn de la propriété Trace de la classe Page, qui seront automatiquement ignorées si l'option Trace est désactivée.

La fin du Response.Write(...)

Le mécanisme de trace constitue en quelque sorte la version évoluée de la technique artisanale utilisée dans le développement ASP, qui consistait à ajouter des instructions `Response.Write` un peu partout dans la page pour dépister un dysfonctionnement.

Activer la trace pour l'ensemble d'une application

Pour activer l'option Trace pour l'ensemble d'une application (et non pas pour une page unique), insérez la balise `<trace enabled=true/>` dans la section `<system.web>` du fichier de configuration de l'application : l'ensemble des requêtes effectuées sera alors consultable via l'URL http://<RacineApplication>/trace.axd.

Voici, par exemple, la marche à suivre pour tracer l'exécution de la page d'accueil de notre étude de cas, en étudiant en particulier le temps de réponse du service Web associé au contrôle `Meteo` développé dans le chapitre précédent :

1 Activez l'option `Trace` pour la page `default.aspx` :

```
<%@ Page Language="<YourLanguage>" Trace="True" %>
```

2 Insérez des instructions de trace (`Trace.Write`) pour encadrer les appels au service Web.

Fichier Meteo.ascx (Version C#)

Ces instructions sont sans effet si l'option `Trace` est désactivée

```
Trace.Write("Contrôle météo","Avant GetWeatherInfos (Paris)");
infos = GetWeatherInfos("LFPO");
Trace.Write("Contrôle météo","Après GetWeatherInfos Paris");
...
Trace.Write("Contrôle météo","Avant GetWeatherInfos (Marseille)");
infos = GetWeatherInfos("LFML");
Trace.Write("Contrôle météo","Après GetWeatherInfos (Marseille)");
```

Si on exécute la page, on constate qu'elle est suivie d'un rapport complet des informations de trace, qui inclut les instructions spécifiées par l'intermédiaire de `Trace.Write` (figure 9-5).

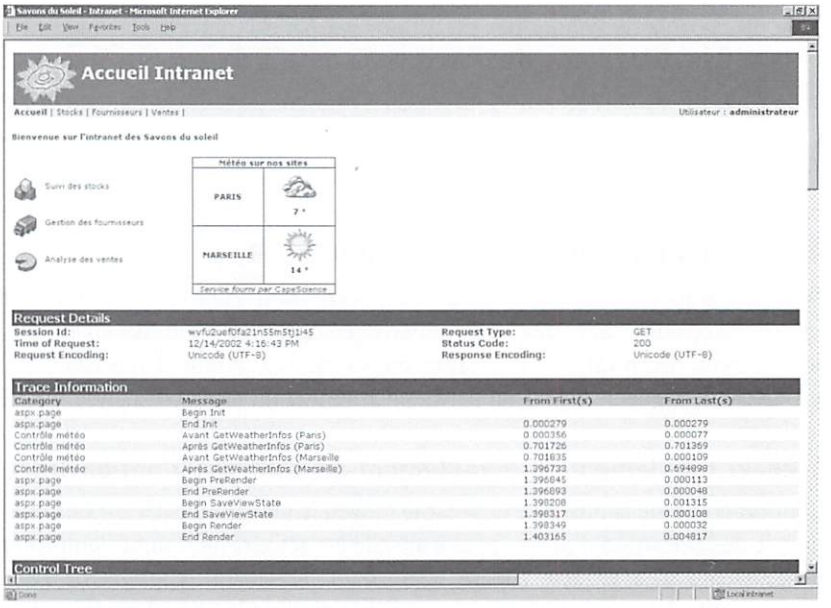


Figure 9–5 Exécution d'une page avec activation de l'option Trace

Si le mécanisme de trace permet d'obtenir rapidement un grand nombre d'informations pertinentes, il ne permet pas de détailler pas à pas l'exécution d'une page : il faut pour cela recourir au débogueur.

Mise en place d'analyseurs de performance

Signalons l'existence de la classe `PerformanceCounter` définie dans l'espace de nommage `System.Diagnostics`, qui facilite la mise en place de mécanismes d'analyse des performances d'une application .NET (pourcentage CPU utilisé, nombre d'accès disques, etc.)

Il n'est pas nécessaire de posséder Visual Studio.NET

Le débogueur .NET est disponible en téléchargement gratuit au sein du kit de développement .NET : autrement dit, il n'est pas nécessaire de posséder Visual Studio.NET pour déboguer des applications .NET.

Examiner en détail le déroulement de l'exécution avec le débogueur .NET

Le kit de développement .NET est fourni avec un débogueur gratuit et très puissant, qui possède pratiquement toutes les fonctionnalités du débogueur de Visual Studio.NET (à l'exclusion du Edit and Continue et du débogage d'applications distantes) :

- exécution pas à pas (avec entrée ou non à l'intérieur des fonctions) ;
- points d'arrêts (fixes ou conditionnels, en fonction d'une expression donnée ou de la survenue d'une exception d'un type donné) ;
- examen dynamique de la valeur des variables ;
- examen dynamique de l'état de la pile et des threads en cours d'exécution.

Pour déboguer une page ASP.NET :

1. Démarrez le débogueur (voir remarque).
2. Ouvrez le fichier source correspondant à la page que vous souhaitez déboguer.
3. Choisissez Debug Processes... dans le menu Tools.
4. Sélectionnez le processus `aspnet_wp.exe` et cliquez sur Attacher (voir figure 9-6).
5. Positionnez au moins un point d'arrêt dans le code de la page (par exemple, dans le gestionnaire `Page_Init` ou `Page_Load`) afin de basculer dans le débogueur lors de l'exécution de la page.
6. Enfin, exécutez la page dans un navigateur : le point d'arrêt s'active dans le débogueur (voir figure 9-7).

Ne pas oublier de positionner l'option Debug à True

Pour pouvoir déboguer une page ASP.NET, il faut s'assurer que la page a été compilée en mode Debug, autrement dit que l'option Debug a été positionnée à `True` dans la directive `<%@ Page...%>` (ou au sein du fichier de configuration, si on souhaite appliquer le mode Debug à l'ensemble des éléments d'une application).

Comment démarrer le débogueur .NET ?

Le débogueur est un exécutable nommé DbgCLR installé par défaut dans le répertoire : `\Program Files\FrameworkSDK\GuiDebug`. Comme l'installation ne crée pas de raccourci vers cet exécutable, le plus simple consiste à créer manuellement un raccourci pour le démarrer plus facilement.

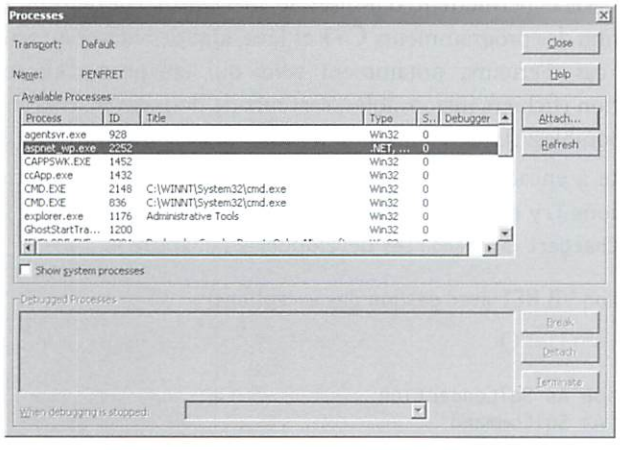


Figure 9–6 Débogage du processus `aspnet_wp.exe`

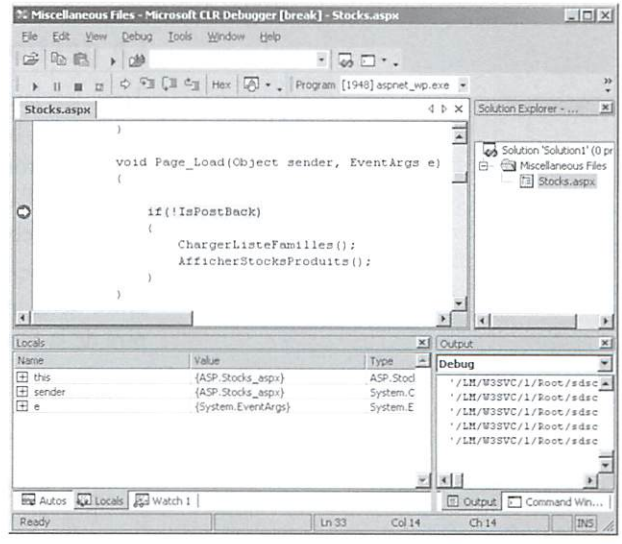


Figure 9–7 Point d'arrêt dans le débogueur

Optimiser les performances de l'application grâce au maintien en cache

ASP.NET offre la possibilité de configurer le maintien en cache d'une page ou d'un contrôle utilisateur en fonction de paramètres divers : délai de conservation dans le cache fixé par l'utilisateur, variation en fonction du type de navigateur utilisé (une nouvelle version de la page est produite pour chaque nouveau navigateur, mais la version en cache est utilisée pour plusieurs requêtes successives émanant d'un même navigateur), de la présence de certains paramètres dans la chaîne de requête ou dans l'en-tête HTTP de la requête.

L'intérêt principal de ce mécanisme est d'optimiser les performances de l'application : par exemple, minimiser le temps d'affichage de la page d'accueil de notre intranet, qui contient le contrôle `Meteo`, lequel interroge le service `GlobalWeather`. En effet, le temps de réponse du service étant de plusieurs secondes et les données fournies n'étant mises à jour que toutes les deux heures, il est pertinent de maintenir en cache le résultat de l'exécution du contrôle.

Pour cela, il suffit d'ajouter la directive suivante en haut de la page `Meteo.ascx` :

```
<%@ OutputCache Duration=7200 %>
```

où `Duration` spécifie le nombre de secondes pendant lequel il faut maintenir la version en cache (ici : deux heures).

D'autres options sont disponibles comme `VaryByCustom`, `VaryByHeader`, `VaryByParam`, qui permettent de paramétrer plus précisément la mise à jour du cache en fonction du navigateur, de l'en-tête HTTP ou de la chaîne de requête, comme précisé précédemment.

Si le débogueur permet d'accélérer la mise au point d'une application, il ne permet pas d'éviter les défaillances d'éléments externes qui conduisent à la génération d'exceptions, dont le traitement est l'objet de la section suivante.

Tableau 9–1 Principales commandes utilisables dans le débogueur

Touche de raccourci	Action
F5	Aller jusqu'au point d'arrêt suivant
F9	Créer ou supprimer un point d'arrêt
F10	Exécuter pas à pas sans entrer dans les fonctions appelées
F11	Exécuter pas à pas en entrant dans les fonctions appelées
Shift + F11	Sortir d'une fonction appélée

La gestion des exceptions

Tous les langages .NET permettent d'utiliser le mécanisme de gestion des exceptions, bien connu des programmeurs C++ et Java, afin de réaliser un traitement de tous les cas d'erreurs, notamment ceux qui émanent d'éléments externes à l'application (fichiers introuvables, serveurs de données inaccessibles, services Web indisponibles...).

Le principe consiste à encadrer le code susceptible de générer des exceptions entre deux instructions `Try` et `Catch`, comme le montre l'exemple qui suit, inspiré de la fonction `ChargerListeFamilles` développée au chapitre 4.

Stocks.aspx – (version VB.NET avec gestion des exceptions)

```
Sub ChargerListeFamilles()

   Dim myConnection As SqlConnection
   Dim myCommand As SqlCommand
   Dim myReader  As SqlDataReader
   Dim SQL As String
```

```
    Try

        myConnection = CType(Session("myConnection"),SqlConnection)
        SQL = "SELECT * FROM FamilleProduit"
        myCommand = new SqlCommand(SQL,myConnection)
        myReader = myCommand.ExecuteReader()
        ListeFamilles.DataSource      = myReader
        ListeFamilles.DataValueField = "ID_FamilleProduit"
        ListeFamilles.DataTextField  = "NomFamille"
        ListeFamilles.DataBind()
        myReader.Close()
    Catch e As Exception

        Response.Redirect("Erreur.aspx?message="+e.Message)

    End Try

End Sub
```

Les instructions susceptibles de générer des exceptions doivent être contenues entre deux instructions `Try` et `Catch`. Si une exception survient, l'exécution du bloc `Try` s'interrompt et on passe directement à l'exécution du bloc `Catch`, lequel peut être optionnellement suivi d'un bloc `Finally` exécuté dans tous les cas (qu'il y ait eu exception ou non).

Cette instruction traite tous les types d'exceptions. Il est également possible de préciser spécifiquement le type des exceptions que l'on souhaite traiter (par exemple : `DataException`), voire de créer plusieurs blocs `Catch` pour gérer différemment plusieurs types d'exceptions. En l'occurrence, notre gestionnaire redirige l'utilisateur vers une page d'erreur spécifique qui affiche le message correspondant à l'exception.

Notons que le développeur a également la possibilité de générer des exceptions à l'aide du mot-clé `Throw`, voire de définir ses propres types d'exceptions, dérivés de la classe `ApplicationException` définie dans l'espace de nommage `System`.

Tel qu'il a été décrit, ce mécanisme présente l'inconvénient de nécessiter l'implémentation d'un bloc `Try/Catch` dans toutes les fonctions susceptibles de générer une exception. Heureusement, nous allons voir qu'ASP.NET dispose également de techniques qui permettent de traiter de manière générique toutes les erreurs pouvant survenir dans une application.

Gestion spécifique des erreurs

Parmi les types d'erreurs susceptibles de survenir lors de l'exécution d'une application Web, on peut distinguer :

- les erreurs liées au serveur HTTP (par exemple : erreur 404 lorsque l'utilisateur demande un fichier inexistant) ;
- les erreurs applicatives (par exemple : indisponibilité d'un serveur de données interrogé par l'application).

ASP.NET propose des solutions techniques pour gérer ces différents cas d'erreurs : il est possible de spécifier une page d'erreur personnalisée associée à un type d'erreur HTTP ainsi que d'intercepter l'ensemble des erreurs applicatives au sein d'un gestionnaire d'événement `Application_Error`, afin de leur appliquer un traitement spécifique.

Spécifier une page d'erreur personnalisée

Pour spécifier une page d'erreur par défaut qui permette de notifier les dysfonctionnements de manière conviviale à l'utilisateur, il suffit d'insérer une section `<customErrors>` dans le fichier de configuration de l'application, en spécifiant :

- le nom de la page d'erreur à l'aide de l'attribut `defaultRedirect` ;

Figure 9–8 Gestion personnalisée des erreurs HTTP 404

- le mode d'affichage des erreurs (`On` : activé ; `Off` : désactivé ; `RemoteOnly` : activé uniquement pour les utilisateurs distants) ;
- les éventuelles pages spécifiques à utiliser pour certains codes d'erreurs HTTP (par exemple : 404 pour les fichiers non trouvés, voir figure 9-8), au sein de sous-sections `<error>` indiquant le code d'erreur (`statusCode`) et la page associée (`redirect`).

Fichier web.config

```
<system.web>
   <customErrors defaultRedirect="Erreur.aspx" mode="On">
     <error statusCode="404" redirect="Erreur404.aspx"/>
   </customErrors>
</system.web>
```

Gérer les erreurs au niveau de la page

Le gestionnaire `Page_Error` permet d'effectuer le même type de traitement au niveau d'une page donnée.

Intercepter l'ensemble des erreurs survenant dans une application

Le gestionnaire `Application_Error` défini dans le fichier `global.asax` permet d'intercepter l'ensemble des erreurs susceptibles de survenir lors de l'exécution d'une application.

Il est possible, par exemple, d'envoyer un message contenant la description de l'erreur à l'administrateur, tout en redirigeant l'utilisateur vers une page conviviale, comme le montre l'exemple de code proposé.

Fichier global.asax (version C#)

```
void Application_Error(Object sender, EventArgs E)
{
   MailMessage mail = new MailMessage();

   mail.From = "webmaster@savonsdusoleil.com";
   mail.To = "<VotreAdresse>";
   mail.Subject = "Une erreur ASP.NET s'est produite";
   mail.Body = Server.GetLastError().ToString();
   mail.BodyFormat = MailFormat.Text;

   SmtpMail.SmtpServer = "<VotreServeurSMTP>";
   SmtpMail.Send(mail);
   Server.ClearError();

   Response.Redirect("Erreur.aspx");
}
```

On prépare un message, destiné à l'administrateur de l'application, contenant la description de l'erreur récupérée grâce à la méthode `GetLastError` de l'objet `Server` (pour plus d'informations sur la classe `MailMessage`, voir le chapitre 6).

On indique que l'erreur a été traitée en appelant la méthode `ClearError` de la classe `Server`.

On redirige explicitement l'utilisateur vers une page d'erreur (NB : l'implémentation de `Application_Error` rend inactif le mécanisme des `<customErrors>` décrit au paragraphe précédent).

Après ce tour d'horizon des possibilités d'ASP.NET en matière de gestion d'erreurs, nous allons maintenant détailler le déploiement d'une application vers un serveur de production.

Déploiement d'une application ASP.NET

En pratique, une application ASP.NET est composée de deux types d'éléments :

- des fichiers source (pages `.aspx` ou `.asmx`, fichiers de configuration `web.config` et `global.asax`, ressources graphiques, etc.) ;
- des assemblages (compilés sous la forme de DLL) auxquels l'application fait référence.

Le mécanisme le plus simple pour déployer une application ASP.NET consiste à effectuer une simple copie des fichiers vers le répertoire de l'application sur la machine cible (les assemblages doivent être copiés dans le sous-répertoire `bin`).

Cette technique, présentée figure 9-9, présente deux avantages majeurs :

- le déploiement peut se faire entièrement par FTP (alors qu'auparavant, les composants ActiveX/COM devaient être enregistrés sur la machine cible via `regsvr32`, ce qui nécessitait généralement l'intervention d'un administrateur) ;
- il existe une séparation nette entre les applications, ce qui élimine tout risque de conflits entre assemblages.

Hébergement gratuit ASP.NET

Un service d'hébergement gratuit d'applications ASP.NET (comprenant 20 Mo de fichiers et 10 Mo dans une base SQL Server) est proposé à l'adresse :

- france.webmatrixhosting.net

Déploiement XCOPY

Cette technique de déploiement par simple copie de fichier est connue sous le nom de « Déploiement XCOPY », par référence à l'utilitaire XCOPY qui permet d'effectuer des copies de fichiers en mode Ligne de commande

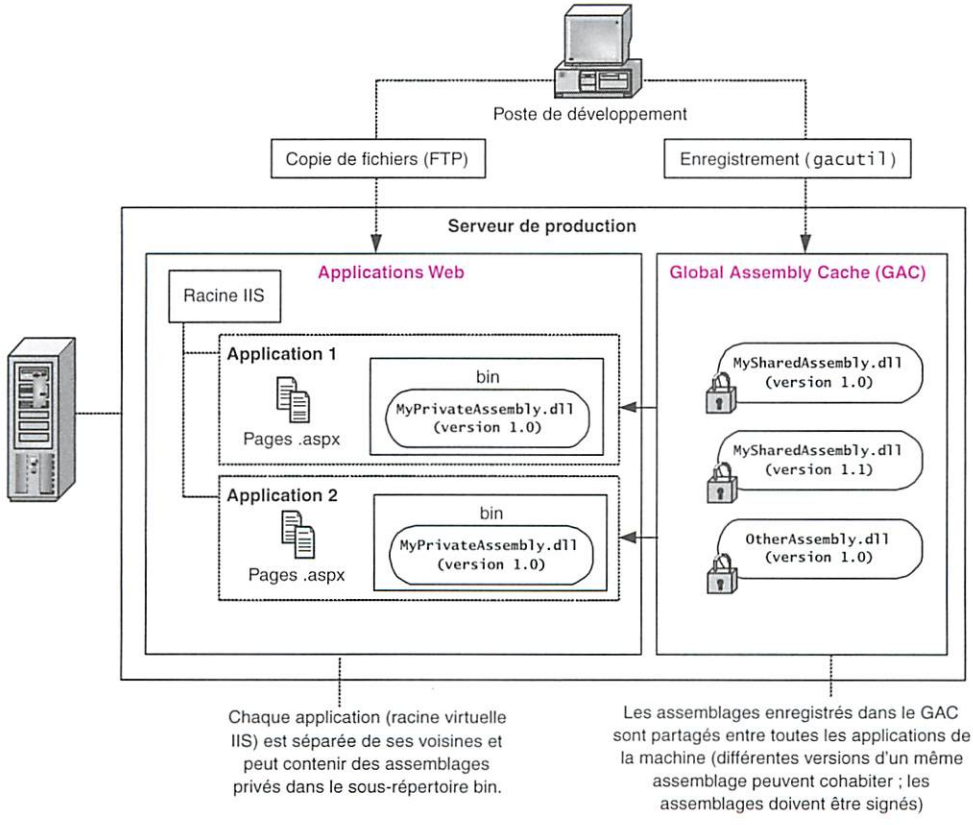


Figure 9–9 Déploiement d'une application ASP.NET

Global Assembly Cache

Le Global Assembly Cache (GAC) est un emplacement utilisé pour le stockage des assemblages partagés entre toutes les applications .NET d'une machine donnée.

Assemblages vs composants COM/ActiveX

Lors du déploiement de composants COM/ActiveX (pour une application ASP classique), le développeur était contraint de partager tous les éléments installés. Désormais, il peut opter selon ses besoins pour un déploiement privé (qui présente, en plus, l'avantage de se résumer à une simple copie de fichier) ou partagé (installation dans le Global Assembly Cache).

À propos de la signature numérique

Un système de cryptographie à clé publique repose sur l'utilisation de deux valeurs : une clé privée utilisée pour crypter les informations et connue de l'émetteur seul, une clé publique qui peut décrypter uniquement les informations cryptées avec la clé privée associée. Un tel système permet de garantir l'identité du signataire, à condition, bien entendu, de garder secrète la clé privée !

Sécuriser ses échanges électroniques avec une PKI, T. Autret *et al.*, Eyrolles 2002

Autres informations associées à l'assemblage

Les deux options présentées (utilisation de l'utilitaire `al.exe` ou spécification des attributs de l'assemblage depuis un fichier source) peuvent s'appliquer non seulement à la signature, mais encore à l'ensemble des informations relatives à l'assemblage : numéro de version, nom, copyright, etc. Pour plus d'informations sur les attributs disponibles, consulter la documentation de l'espace de nommage `System.Reflection`.

Dans certains cas, il peut être utile de partager des assemblages fréquemment utilisés entre toutes les applications d'une machine donnée (afin d'éviter d'avoir à les déployer dans le répertoire `bin` de chaque application) : c'est le rôle du *Global Assembly Cache* (GAC), dont nous allons maintenant décrire le fonctionnement.

Déployer un assemblage dans le Global Assembly Cache

Avant de pouvoir être installé dans le *Global Assembly Cache*, un assemblage doit être signé numériquement à l'aide d'un mécanisme clé publique/clé privée, ceci pour deux raisons :

- la clé publique sert d'identifiant unique pour l'assemblage, évitant ainsi les collisions de noms (plusieurs assemblages peuvent porter le même nom, à partir du moment où leurs clés publiques sont différentes) ;
- le mécanisme de cryptage assure l'intégrité de l'assemblage (si l'assemblage original est remplacé à des fins malveillantes, le nouvel assemblage ne pourra pas être décrypté par la clé publique, car il aura été signé avec une clé privée différente de la clé originale) ;
- La signature d'un assemblage commence toujours par la génération d'une paire de clés à l'aide de l'utilitaire `sn.exe` (pour *strong name*), dont la syntaxe d'utilisation est la suivante :

```
sn -k <NomFichier>
```

où `<NomFichier>` désigne le nom du fichier de clés à générer (auquel on donne généralement l'extension `.snk`).

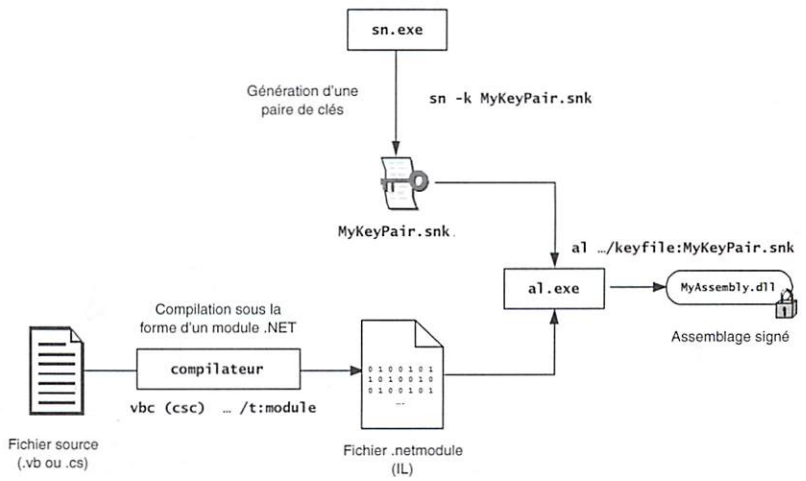


Figure 9–10 Une des techniques possibles pour la signature d'un assemblage

Gestion des différences de langue entre serveurs

Un problème classique dans le déploiement d'applications est la gestion des différences de langue entre le serveur de développement et le serveur de production, lesquelles peuvent avoir un impact sur le formatage des valeurs dépendant des us et coutumes liés à la langue : par exemple, le contrôle serveur `Calendar` utilisé pour la saisie des nouvelles commandes (voir chapitre 5) sera par défaut affiché en anglais sur un serveur installé en anglais.

Il est heureusement très facile de spécifier une langue pour l'application (par exemple, le français) qui soit différente de la langue du système d'exploitation du serveur ; il suffit pour cela d'insérer une balise `<globalization>` qui spécifie la langue souhaitée dans le fichier de configuration de l'application (figure 9-11) :

```
<globalization culture="fr-FR" />
```

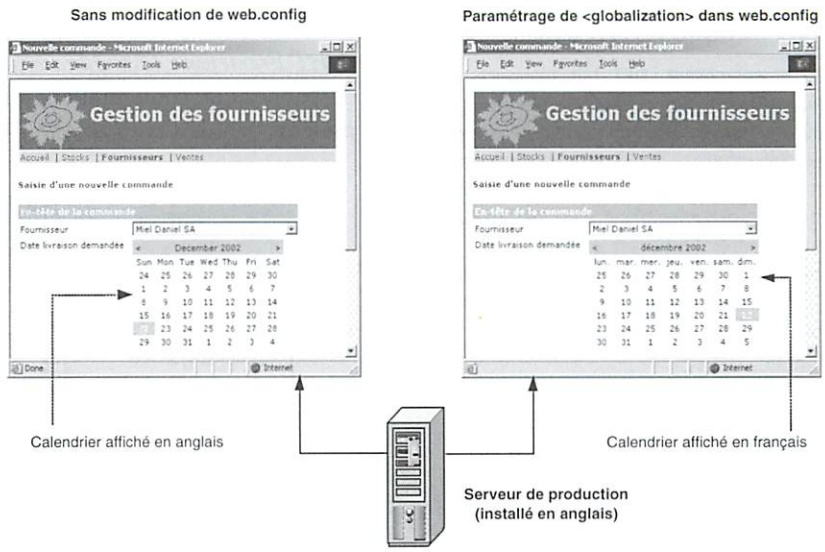


Figure 9–11 Gestion de l'internationalisation

Une fois la paire de clés générée, deux techniques sont disponibles pour signer un assemblage :

- utiliser l'utilitaire `al.exe` (Assembly Linker) pour lier la clé à un ou plusieurs modules .NET (voir figure 9-10) ;
- faire référence à la clé depuis le code source, afin que la signature soit automatiquement effectuée lors de la compilation : pour cela, il faut ajouter aux sources de l'assemblage un fichier nommé, par exemple, `AssemblyInfo.cs` ou .vb, contenant l'instruction suivante :

```
<Assembly: AssemblyKeyFile("MyKeyPair.snk")> (Version VB.NET)
[assembly: AssemblyKeyFile("MyKeyPair.snk")] (Version C#)
```

Une fois l'assemblage signé, il peut être installé dans le *Global Assembly Cache* à l'aide de l'utilitaire `gacutil`, dont la syntaxe d'utilisation est la suivante (voir figure 9-12) :

```
gacutil -i <NomAssemblage>
```

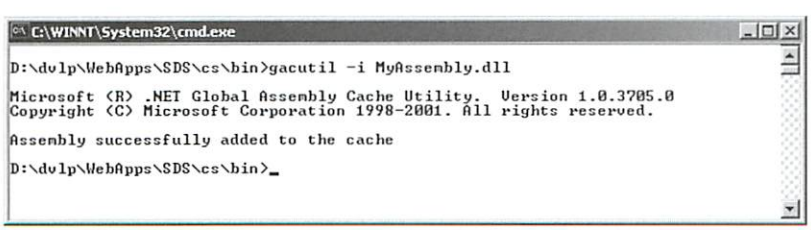


Figure 9–12 Installation d'un assemblage dans le Global Assembly Cache

Droits spécifiques nécessaires

Contrairement au déploiement d'assemblages privés (XCOPY), l'installation d'un nouvel assemblage partagé dans le GAC nécessite d'avoir des droits spécifiques sur la machine cible (par exemple, accès en mode Terminal Server pour pouvoir exécuter `gacutil`).

Localisation du Global Assembly Cache

Par défaut, le Global Assembly Cache est installé dans le répertoire `\WINNT\assembly`.

Ordre de recherche des assemblages

Lorsqu'un composant .NET fait référence à un assemblage non signé, seul le répertoire des assemblages privés de l'application est examiné ; s'il est fait référence à un assemblage signé, le répertoire des assemblages privés est examiné en priorité puis, si l'assemblage n'est pas trouvé, le Global Assembly Cache est examiné à son tour. Notons au passage que plusieurs versions différentes d'un même assemblage peuvent cohabiter dans le Global Assembly Cache.

L'installation de l'assemblage peut être contrôlée grâce à l'utilitaire *Microsoft .NET Framework Configuration*, accessible via le dossier *Outils d'administration* du panneau de configuration, qui permet de visualiser l'ensemble des éléments présents dans le *Global Assembly Cache* (voir figure 9-13).

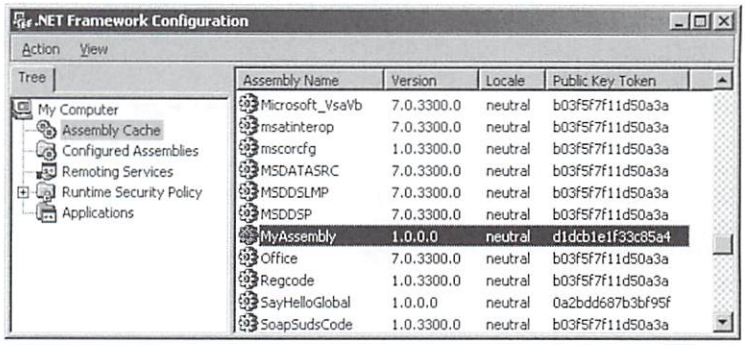


Figure 9–13 Visualisation du contenu du Global Assembly Cache

En résumé...

Dans ce dernier chapitre, nous avons passé en revue les principaux sujets relatifs à la sécurisation, la configuration, l'optimisation et la mise en production d'une application ASP.NET :

- authentification des utilisateurs et gestion spécifique des autorisations ;
- sécurisation de l'appel à un service Web ;
- rôle des fichiers de configuration (`machine.config` et `web.config`) ;
- analyse de l'exécution d'une application à l'aide du mécanisme de trace ;
- optimisation des performances à l'aide du cache ;
- utilisation du débogueur ;
- gestion des exceptions et traitement spécifique des erreurs ;
- spécification de la langue de l'application ;
- déploiement d'une application et rôle du *Global Assembly Cache*.

En conclusion...

Tout au long de cette présentation d'ASP.NET, nous avons eu l'occasion d'illustrer les principales possibilités de cette nouvelle technologie et les évolutions majeures qu'elle apporte au développement Web :

- une interface Web peut désormais être conçue comme un assemblage de contrôles graphiques au lieu d'un mélange de contenu HTML et de scripts serveur : la séparation entre code et contenu graphique augmente la lisibilité des applications et facilite leur maintenance, tandis que l'encapsulation au sein de contrôles accroît la productivité du développement ;
- le modèle de programmation événementielle, avec gestion automatique des allers-retours vers le serveur (`PostBack`) et conservation de l'état des contrôles (`ViewState`) permet d'implémenter des pages Web plus interactives, sans pour autant nécessiter le recours à des scripts client fastidieux ;
- le caractère compilé des pages améliore la performance des applications et l'utilisation de langages objet augmente la robustesse du code (encapsulation, typage fort) et multiplie les possibilités de réutilisation (héritage, redéfinition de méthodes) ;
- la bibliothèque ADO.NET et les contrôles serveur liés aux données (listes, grilles) permettent d'accéder à des sources de données en tenant compte du caractère déconnecté des applications Web et d'implémenter facilement la consultation, la mise à jour ou la suppression de données, ainsi que la gestion des transactions.
- les modèles proposés pour le développement de nouveaux composants (contrôles utilisateur, contrôles serveur spécifiques, objets métier) permettent l'extension des possibilités de la bibliothèque standard et favorisent la réutilisation ;
- de nombreuses classes utilitaires rendent aisées la génération et la manipulation de données et de schémas XML ainsi que l'application de transformations XSL ;

- l'architecture est adaptée à la réalisation de services Web, qui peuvent très facilement être implémentés, testés et sécurisés, ainsi qu'à l'appel de services Web distants, incluant notamment la génération automatique de *proxies* et le paramétrage d'en-têtes SOAP.
- les possibilités de configuration sont multiples et simples à mettre à œuvre : gestion spécifique des erreurs (redirection de l'utilisateur vers une page conviviale, envoi de messages à l'administrateur), mécanismes de sécurisation intégrés (redirection automatique des utilisateurs inconnus vers une page d'authentification, gestion des autorisations), optimisation des performances grâce au mécanisme de cache, etc.

Cette première approche ne se prétend pas exhaustive : certains sujets comme les contrôles mobiles, qui permettent de générer du contenu adapté à des terminaux mobiles (WML, cHTML), ou l'interopérabilité avec les objets COM n'ont pas pu être abordés dans le cadre de cet ouvrage.

Souhaitons néanmoins que ce « Cahier du Programmeur » aura permis au lecteur d'appréhender les mécanismes fondamentaux de l'architecture ASP.NET et de se forger une opinion sur cette technologie qui est, sans nul doute, promise à un bel avenir.

Annexes

Fichiers de l'application

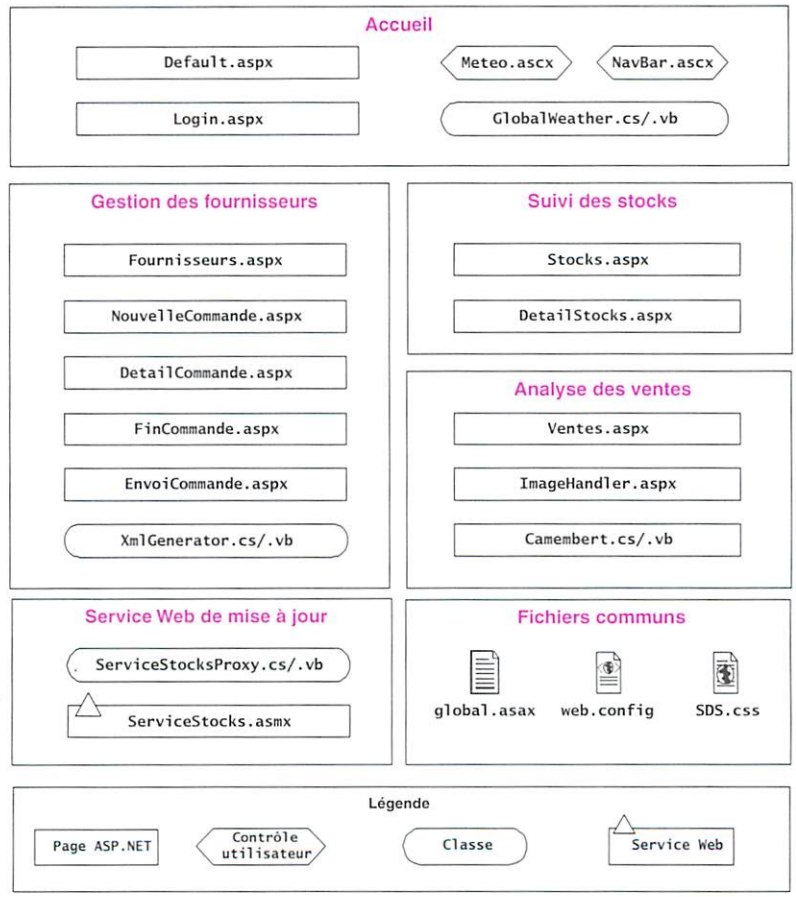


Tableau A–1 Liste des fichiers de l'application

Fichier	Type	Description	Chap.
Camembert.cs/.vb	Classe	Implémentation du contrôle serveur spécifique « Camembert »	7
Default.aspx	Page ASP.NET	Page d'accueil de l'intranet	3
DetailCommande.aspx	Page ASP.NET	Consultation et mise à jour du détail d'une commande	5
DetailStocks.aspx	Page ASP.NET	Consultation de l'historique du stock pour un produit	4
EnvoiCommande.aspx	Page ASP.NET	Envoi de commande à un fournisseur	6
FinCommande.aspx	Page ASP.NET	Page récapitulative affichée à la fin de la saisie d'une commande	6
Fournisseurs.aspx	Page ASP.NET	Affichage de la liste des commandes fournisseur	5
Global.asax	Fichier .asax	Fichier de configuration de l'application	4, 9
GlobalWeather.cs/.vb	Classe	Proxy pour l'appel du service Web GlobalWeather	8
ImageHandler.aspx	Page ASP.NET	Page utilitaire utilisée pour l'affichage du contrôle « Camembert »	7
Login.aspx	Page ASP.NET	Authentification des utilisateurs	9
Meteo.ascx	Contrôle utilisateur	Affichage de la météo à Paris et à Marseille	8
NavBar.ascx	Contrôle utilisateur	Barre de navigation	3, 9
NouvelleCommande.aspx	Page ASP.NET	Saisie d'une nouvelle commande	5
SDS.css	Feuille de style	Feuille de style appliquée aux fichiers de l'application	3
ServiceStocks.asmx	Service Web	Service Web de mise à jour des données des stocks	8, 9
ServiceStocksProxy.cs/.vb	Classe	Proxy pour l'appel du service Web ServiceStocks	8, 9
Stocks.aspx	Page ASP.NET	Consultation des stocks par famille de produits	4
Ventes.aspx	Page ASP.NET	Consultation des ventes par région et par produit	7
web.config	Fichier .config	Fichier de configuration de l'application	4, 9

Structure de la base de données

Rappel

Le script complet de création de la base est téléchargeable sur le site de l'étude de cas :

- www.savonsdusoleil.com

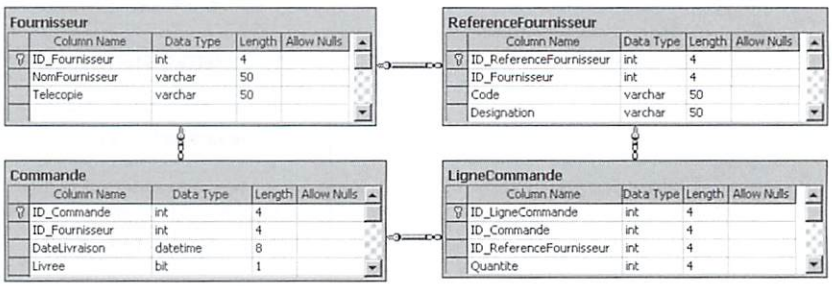


Figure A–1 Schéma de la base (module fournisseur)

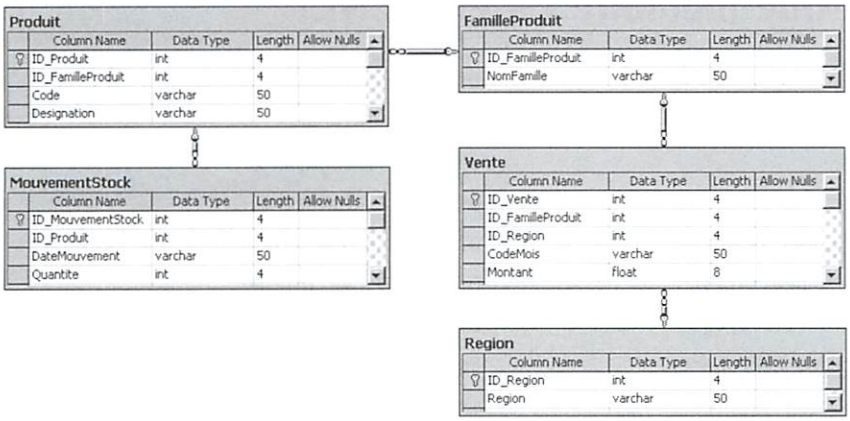


Figure A–2 Schéma de la base (modules stocks et ventes)

Tableau A–3 Liste des tables de la base

Table	Données contenues
Commande	Liste des commandes passées auprès des fournisseurs
FamilleProduit	Liste des familles de produits
Fournisseur	Liste des fournisseurs
LigneCommande	Liste des lignes associées aux commandes fournisseur
MouvementStock	Liste des mouvements de stocks (ventes, approvisionnements)
Produit	Liste des produits vendus par la société
ReferenceFournisseur	Listes des références disponibles auprès des fournisseurs
Region	Liste des régions
Vente	Détail des ventes mensuelles par région et par famille de produit

Tableau A–4 Liste des procédures stockées de la base

Table	Rôle
CreerCommande	Création d'une nouvelle commande fournisseur
DetailsCommande	Informations détaillées relatives à une commande
EtatStock	État des stocks pour une famille de produits donnée
HistoriqueStock	Historique du stock pour un produit donné
ListeCommandes	Liste des commandes fournisseur
ListeLignesCommandes	Liste des lignes pour une commande donnée
ReferenceFournisseur	Listes des références disponibles auprès des fournisseurs
VentesMensuellesParFamille	Répartition des ventes par famille de produit pour un mois donné
VentesMensuellesParRegion	Répartition des ventes par région pour un mois donné

Aide-mémoire C# et VB.NET

Opération	C#	VB.NET
Déclarer une variable	`int i;`	`Dim i As Integer`
Déclarer et initialiser une variable	`int i = 0;`	`Dim i As Integer = 0`
Déclarer un objet	`Object o;`	`Dim obj As Object`
Déclarer et allouer un objet	`Object o = new Object();`	`Dim obj As New Object`
Déclarer un tableau	`int tab[] = new int[3];`	`Dim tab(3) As Integer`
Déclarer et initialiser un tableau	`int tab[] = {1,2,3} ;`	`Dim tab As Integer = {1,2,3}`

Déclarer une propriété	`public String MyProperty`
	`{`
	`  get { ... return ... ;}`
	`  set { ....= value; }`
	`}`
		`Public Property Name As String`
		`  Get`
		`  ...`
		`  Return ...`
		`  End Get`
		`  Set`
		`  ... = Value`
		`  End Set`
		`End Property`

Accéder à une propriété indexée	`s = Request.QueryString["id"];`	`s = Request.QueryString("id")`

Énumérer une collection	`foreach ( String s in coll )`
	`{`
	`...`
	`}`
		`Dim s As String`
		`For Each s In Coll`
		`...`
		`Next`

Utiliser une boucle `for`	`for (int i=0; i<3; i++)`
	`{`
	`...`
	`}`
		`Dim I As Integer`
		`For I = 0 To 2`
		`...`
		`Next`

Utiliser une boucle `while`	`int i = 0;`
	`while (i<3)`
	`{`
	`...`
	`i++;`
	`}`
		`Dim I As Integer`
		`I = 0`
		`Do While I < 3`
		`...`
		`I += 1`
		`Loop`

Utiliser un aiguillage `select/case`	`switch (i)`
	`{`
	`case 1 :`
	`...`
	`break;`
	
	`case 2 :`
	`...`
	`break;`
	
	`default:`
	`...`
	`break;`
	`}`
		`Select Case i`
		`Case 1`
		`...`
		`Case 2`
		`...`
		`Case Else`
		`...`
		`End Select`

Opération	C#	VB.NET

Utiliser un aiguillage `if/else`	`if (a == b)`
	`{`
	`  ...`
	`}`
	`else`
	`{`
	`  ...`
	`}`
		`If (a = b) Then`
		`  ...`
		`Else`
		`  ...`
		`End If`

Déclarer une méthode	`void f()`
	`{`
	`  ...`
	`}`
	
	`int g(int a)`
	`{`
	`  ...`
	`  return i;`
	`}`
		`Sub f()`
		`  ...`
		`End Sub`
		
		`Function g(a As Integer) _`
		`As Integer`
		`  ...`
		`  Return i`
		`End Function`

Gérer les exceptions	`try`
	`{`
	`  ...`
	`}`
	`catch(Exception e)`
	`{`
	`  ...`
	`}`
	`finally`
	`{`
	`  ...`
	`}`
		`Try`
		`  ...`
		`Catch e As Exception`
		`  ...`
		`Finally`
		`  ...`
		`End Try`

Effectuer une conversion de type	`c =(SqlConnection)Session["c"]`	`c=CType(Session("c"),_` `SqlConnection)`
Insérer un commentaire	`// Ceci est un commentaire`	`' Ceci est un commentaire`
Faire référence à NULL	`if (o == null) ...`	`if (o = Nothing) ...`
Importer un espace de nommage	`using System;`	`Imports System`
Spécifier un attribut	`[WebMethod(EnableSession=True)]`	`<WebMethod(EnableSession:=True)>`

Déclarer une classe	`namespace N`
	`{`
	`  public class C : Base`
	`  {`
	`  public C()`
	`  {`
	`  ... // Constructeur`
	`  }`
	`  ...`
	`  }`
	`}`
		`Namespace N`
		
		`Public Class C : Inherits Base`
		`  Public Sub New()`
		`  ... 'Constructeur`
		`  End Sub`
		`  ...`
		`End Class`
		
		`End Namespace`

Compiler une classe	`csc /out:Output.dll /t:library` `➥ MyClass.cs /r:[references]`	`vbc /out:Output.dll /t:library` `➥ MyClass.vb /r:[references]`

Liens utiles

Nom	Description	URL
15 seconds	Articles, tutoriels et liens relatifs .NET (en anglais)	www.15seconds.com
ASP Today	Articles relatifs à ASP et ASP.NET (en anglais)	www.asptoday.com
ASP.NET	Site officiel d'ASP.NET	www.asp.net
ASP.NET Pro	Forums, news, articles relatifs à ASP.NET (en anglais)	www.aspnetpro.com
ASP-PHP	Site dédié à ASP, ASP.NET et PHP (en français)	www.asp-php.net
C# Today	Site spécialisé sur le langage C# (en anglais)	www.csharptoday.com
Code Guru	Articles et news relatifs à .NET (en anglais)	www.codeguru.com
Code Project	Articles et news relatifs à .NET (en anglais)	www.codeproject.com
Component Source	Catalogue de composants complémentaires pour ASP. NET	www.componentsource.com
CSharpFr	Site dédié au langage C# (en français)	www.csharpfr.com
DNZone	Articles, tutoriels et liens relatifs .NET (en anglais)	www.dnzone.com
DotNetFr	Site communautaire pour les développeurs .NET (en français)	www.dotnet-fr.org
GASP	Ressources techniques relatives à ASP.NET (en français)	www.gasp.fr
GotDotNet	Site communautaire pour les développeurs .NET (en anglais)	www.gotdotnet.com
Learn ASP	Ressources techniques relatives à ASP.NET (en anglais)	www.learnasp.com
Master C#	Articles et ressources relatifs à C# et ASP.NET (en anglais)	www.mastercsharp.com
MSDN	Documentation en ligne de la bibliothèque .NET	msdn.microsoft.com
Savons du Soleil	Site de l'étude de cas de cet ouvrage	www.savonsdusoleil.com
Visual Map	Composants cartographiques pour ASP.NET	www.visualmap.net
Web Matrix Hosting	Hébergement gratuit ASP.NET	france.webmatrixhosting.net

Index

Dépôt légal : Février 2003
N° d'éditeur : 6864
Imprimé en France

Achevé d'imprimer le 17 janvier 2003
sur les presses de l'imprimerie «La Source d'Or»
63200 Marsat
Imprimeur n° 11208