les Cahiers du Programmeur

# ASP.NET

## Infrastructure Web d'une PME avec ASP.NET

Thomas Petillon

EYROLLES

Thomas **Petillon**

# les Cahiers du Programmeur ASP.NET

© Groupe Eyrolles, 2003

ISBN : 2-212-11210-6

diffusé sous licence Creative Commons by-nc-nd 2.0

EYROLLES

# Avant-propos

## Quel est l'objectif de cet ouvrage ?

Les développeurs Web ont probablement tous entendu parler d'ASP.NET. Néanmoins, nombreux sont ceux qui hésitent encore à franchir le pas vers cette nouvelle technologie, qui constitue pourtant une évolution majeure pour la réalisation des applications Web :

- les pages Web compilées et implémentées en langage objet apportent des améliorations notables en termes de performance et robustesse ;
- l'utilisation de contrôles graphiques encapsulant la génération du HTML augmente drastiquement la productivité du développement et les possibilités de réutilisation ;
- le mécanisme de gestion des événéments permet d'implémenter plus facilement des pages Web interactives, sans avoir recours à des scripts clients ;
- concepteurs graphiques et développeurs peuvent travailler simultanément sur le même projet, grâce à un mécanisme de séparation de la présentation et du code ;
- un ensemble de mécanismes applicatifs fournis en standard (sécurité, gestion des erreurs, internationalisation) facilite la tâche au développeur ;
- le déploiement des applications Web peut désormais s'effectuer par simple copie de fichier, sans nécessiter l'enregistrement de composants sur le serveur ;
- enfin, la bibliothèque du framework .NET offre un nombre imposant de classes utilitaires : accès à des sources de données, gestion de documents XML, mise en œuvre de services Web, etc.

Ce livre a pour objet de présenter de manière pragmatique le champ des nouvelles possibilités offertes par ASP.NET à travers une étude de cas : la mise en place d'une infrastructure d'échanges de données via le Web pour une PME spécialisée dans la vente par correspondance.

# À qui s'adresse cet ouvrage ?

Cet ouvrage s'adresse à tous les lecteurs désireux de découvrir la technologie ASP.NET :

- aux développeurs Web, utilisateurs d'ASP, PHP, ColdFusion ou JSP, qui vont voir leurs possibilités décuplées ;
- aux développeurs habitués aux environnements client-serveur (Visual Basic, C++, Java…) qui vont enfin pouvoir effectuer des développements Web avec un niveau de puissance conforme à leurs habitudes ;
- aux chefs de projets désireux d'avoir une vision générale et pragmatique de ce que permet ASP.NET ;
- d'une manière générale, à tous les lecteurs curieux de découvrir les possibilités offertes par cette nouvelle technologie.

**Exemples de code**

Les exemples de code sont présentés dans deux langages : C# et VB.NET.

Pour profiter au mieux des exemples présentés, il est préférable d'avoir des notions sur le développement Web et les bases de données. En revanche, la connaissance des langages VB.NET et C#, utilisés lors de l'étude de cas, n'est pas préalablement requise : on présente en annexe un résumé rapide des principales règles syntaxiques de ces langages, qui sont facilement assimilables par tout développeur pratiquant déjà un langage de programmation.

## Configuration logicielle requise

**Choix des outils**

Les lecteurs qui le souhaitent pourront, s'ils le préfèrent, utiliser Visual Studio.NET à la place de Web Matrix et Microsoft Access ou SQL Server à la place de MSDE.

Pour pouvoir reproduire les exemples présentés dans cet ouvrage, la configuration logicielle suivante est nécessaire :

- Windows 2000 ou Windows XP (toutes versions) ;
- ASP.NET (téléchargement gratuit sur www.asp.net) ;
- environnement de développement Web Matrix (gratuit sur www.asp.net) ;
- moteur de base de données MSDE (gratuit sur www.asp.net).

# Sujets couverts par cet ouvrage

Le fil conducteur de cet ouvrage est la réalisation d'un intranet pour une PME.

*La société « Les Savons du Soleil », petite entreprise spécialisée dans la vente par correspondance de produits cosmétiques, est répartie géographiquement sur plusieurs sites et éprouve de ce fait des difficultés dans ses échanges d'informations au quotidien.*

*Elle a par conséquent décidé de mettre en place une base de données centralisée, accessible via une interface Web et mise à jour automatiquement depuis l'usine marseillaise afin de permettre aux départements logistiques et marketing, situés à Paris, de travailler plus efficacement.*

*Elle a choisi d'effectuer ces développements avec la technologie ASP.NET qui permet de couvrir tous ses besoins, de la consultation et la mise à jour de la base aux échanges XML avec les fournisseurs, en passant par la sécurisation des accès et l'implémentation d'un service Web de mise à jour de la base.*

Cette étude de cas sera l'occasion de faire un très large tour d'horizon des possibilités offertes par la technologie ASP.NET :

- dans le chapitre 1, nous effectuons une présentation détaillée de l'étude de cas et nous exposons les justifications techniques du choix d'ASP.NET, de la base de données MSDE et de l'environnement de développement Web Matrix ;
- le chapitre 2 décrit l'installation des outils de développement et la mise en place de la base de données utilisée par l'étude de cas, en présentant au passage l'interface de gestion des données de Web Matrix ;
- le chapitre 3 est consacré à l'architecture d'une page ASP.NET : à travers la réalisation du squelette de l'application, nous illustrons la séparation du code et de la partie graphique (avec les deux variantes *code behind* ou *code in-line*), le rôle des contrôles serveur et la notion de contrôle utilisateur ;
- dans le chapitre 4, consacré au module de suivi des stocks, nous étudions comment accéder à des données de la base en lecture en utilisant la bibliothèque ADO.NET et le contrôle serveur `DataGrid`, dont on souligne au passage les nombreuses possibilités de paramétrage ; le mécanisme de gestion des événements d'ASP.NET et la notion de conservation de l'état (`ViewState`) sont également décrits ;
- le chapitre 5 est consacré au module de gestion des commandes fournisseur, lequel nécessite non seulement un accès en lecture à la base (qui sera cette fois mis en œuvre à l'aide du contrôle `Repeater`) mais aussi un accès en écriture (modification et ajout de commandes), qui sera implémenté à l'aide des classes `SqlDataAdapter` et `DataSet` ; nous abordons également dans ce chapitre les contrôles de validation et la gestion des transactions ;
- ce module est enrichi dans le chapitre 6 avec l'implémentation d'échanges XML avec les fournisseurs, qui permet d'aborder quelques-unes des nombreuses possiblités d'ASP.NET sur le sujet : génération de fichiers XML, application d'une transformation XSL, envoi de fichiers par messagerie ;
- le chapitre 7 présente la notion de contrôle serveur spécifique à travers l'implémentation du module d'analyse des ventes : la répartition des ventes par régions est représentée graphiquement à l'aide d'un composant du marché ; puis, un contrôle serveur de type graphique composé de secteurs colorés est implémenté pour afficher les ventes par famille de produits ;
- dans le chapitre 8, nous réalisons un service Web permettant la mise à jour des données de stocks depuis un système externe, puis nous implémentons un contrôle utilisateur qui réalise l'affichage des informations météorologiques fournies par un service Web externe ;
- enfin, le chapitre 9 passe en revue les aspects liés à la mise en production de l'application : sécurisation, configuration, internationalisation, analyse des performances, gestion spécifique des erreurs et déploiement.

**Étude de cas en ligne**

La version en ligne et les sources de l'étude de cas sont disponibles sur les sites d'accompagnement aux adresses :

- http://www.savonsdusoleil.com
- http://www.editions-eyrolles.com

Les lignes de code réparties sur plusieurs lignes en raison de contraintes de mise en pages sont signalées par la flèche ➥.

Les chapitres étant relativement indépendants, le lecteur qui le souhaite peut aborder directement les sujets qui l'intéressent, bien qu'il soit conseillé d'avoir préalablement lu la présentation de l'étude de cas et mis en place les outils de développement et la base de données.

Le chef de projet tirera probablement avantage de la lecture de l'ensemble des chapitres, même s'il ne rentre pas dans le détail des codes proposés ; en outre, un résumé de synthèse est systématiquement inclus à la fin de chaque chapitre.

À l'issue de cette étude de cas, le lecteur aura mis en œuvre une application concrète qui lui aura permis – du moins, nous l'espérons – d'acquérir une bonne vision d'ensemble des possibilités d'ASP.NET et des mécanismes fondamentaux de cette technologie.

## Remerciements

Je tiens à remercier toutes les personnes qui ont rendu possible la parution de cet ouvrage et avec qui ce fut un réel plaisir de travailler : Aurélie, pour la pertinence de ses relectures, Sophie et Anne, pour leurs précieux conseils et leur efficacité, Martine et Jean-Marie, pour le formidable travail réalisé et enfin Muriel, qui a réussi à me convaincre de tenter l'aventure !

Thomas PETILLON

petillon@topic.fr

www.topic.fr

# Table des matières


**AVANT-PROPOS** ..... V

**Quel est l'objectif de cet ouvrage ? V**
**À qui s'adresse cet ouvrage ? VI**
**Sujets couverts par cet ouvrage VI**
**Remerciements VIII**

**1. L'ÉTUDE DE CAS « LES SAVONS DU SOLEIL »** ..... 1

**Une PME géographiquement dispersée 2**
**Des échanges d'informations fastidieux entre les sites 2**
Inconvénients liés à la situation actuelle 3
**Le projet de nouvelle infrastructure technique 4**
Mise en place d'une base de données centralisée 4
Données nécessaires à l'analyse des ventes et au suivi des stocks 4
Données nécessaires à la gestion des commandes fournisseur 5
Interface Web pour la consultation/mise à jour de la base 6
Module de suivi des stocks 6
Module de gestion des commandes fournisseur 7
Module d'analyse des ventes 7
Authentification 8
Mise à jour automatique des données de la base 9
**Choix d'architecture technique : ASP.NET, MSDE et Web Matrix 10**
Choix de la technologie de développement Web 10
Qu'est-ce qu'ASP.NET ? 11
Les nouveautés apportées par ASP.NET 13
Choix de la base de données 14
MSDE : une version allégée et gratuite de SQL Server 2000 15
Choix de l'environnement de développement 15
Web Matrix : un environnement de développement efficace et gratuit 16
**En résumé 16**

**2. INSTALLATION DES OUTILS ET CRÉATION DE LA BASE** ..... 17

**Installation des outils de développement 18**
Obtention du kit de développement .NET 18
Prérequis à l'installation du kit de développement .NET 19
Installation du serveur Web Internet Information Server (IIS) 19
Installation du kit de pilotes de bases de données MDAC 19
Désinstallation des versions beta antérieures 20
Installation du kit de développement 20
Installation de Microsoft SQL Server Desktop Engine (MSDE) 21
Installation des exemples du kit de développement .NET 22
Téléchargement et installation de Web Matrix 22
**Création de la structure de la base de données 22**
Création de la structure de la base avec Web Matrix 23
Création d'une nouvelle base 23
Création de tables 24
Création de procédures stockées 25
Conclusion sur l'interface de gestion de données de Web Matrix 25
Alternative 1 - Utilisation de la console SQL Server Enterprise Manager 26
Alternative 2 - Utilisation de Microsoft Access 27
Alternative 3 - Utilisation de scripts SQL 27
**En résumé… 28**

**3. ARCHITECTURE D'UNE PAGE ASP.NET** ..... 29

**La page Web vue comme une interface classique 30**
Un contenu HTML mieux organisé grâce aux contrôles serveur 31
Un code plus structuré grâce à la séparation du contenu HTML et de la cinématique 33
Alternative 1 - Placer le code et le HTML de la page dans un même fichier 33
Alternative 2 - Placer le code de la page dans un fichier séparé 35
Faciliter la réutilisation avec les contrôles utilisateur 37
**Mise en pratique : réalisation d'une barre de navigation 38**
Création de la barre de navigation 38
Création d'un contrôle utilisateur avec Web Matrix 39
Réalisation de la partie graphique de la barre de navigation 40
Programmation de la barre de navigation 42
Création du squelette de l'intranet 44
Tester l'application 47
**En résumé… 48**

**4. CONSULTER UNE BASE DE DONNÉES : L'INTERFACE DE SUIVI DES STOCKS** ..... 49
**Réalisation de la maquette de l'interface 50**
Maquette de la page de consultation des stocks par famille de produits 50
Maquette de la page de consultation de l'historique du stock d'un produit 52
**Mise en place des liens entre l'interface et la base de données 54**
Présentation de la librairie ADO.NET 54
Établissement de la connexion à la base de données 55
Connexion à la base avec SqlConnection 58
Partage de la connexion à la base avec l'objet Session et le fichier global.asax 58
Liaison du contrôle DropDownList à la table FamilleProduit 61
Alternative 1 - Utilisation de SqlCommand et SqlDataReader 62
Alternative 2 - Utilisation de SqlDataAdapter, DataTable et DataView 63
Tester le remplissage de la liste des familles de produits 65
Utilisation de DataGrid pour afficher l'état des stocks 66
Personnalisation du contrôle DataGrid 68
Implémentation de la page de consultation de l'historique 71
**Rendre la page interactive grâce à la gestion des événements 73**
Mieux structurer le code au sein d'une page grâce aux événements prédéfinis 73
Gérer le changement de famille grâce aux événements déclenchés par DropDownList 74
**En résumé… 77**

**5. METTRE À JOUR UNE BASE DE DONNÉES : LA GESTION DES COMMANDES FOURNISSEUR** ..... 79
**Affichage de la liste des commandes fournisseur 80**
Réalisation de la maquette de la liste des commandes avec Repeater 80
Paramétrage du contrôle Repeater 82
Liaison du contrôle Repeater à une source de données 83
Réalisation de la maquette HTML des éléments du contrôle Repeater 84
Formatage des éléments liés aux données 87
Paramétrage du bouton Détails pour accéder au détail d'une commande 90
Gestion de la suppression d'une commande avec LinkButton 90
Utilisation de l'événement OnItemCreated pour ajouter un message de confirmation 93
**Édition d'une commande existante 94**
Aller plus loin : ajouter un mécanisme de pagination à la liste des commandes 95
Réalisation de la maquette de la page affichant le détail d'une commande 96
Consultation et mise à jour de la commande avec SqlDataAdapter 99
**Ajout d'une nouvelle commande 102**
Réalisation de la maquette de l'interface 103
Implémentation de l'ajout d'une nouvelle commande avec SqlDataAdapter 106
Validation des informations saisies par l'utilisateur 111
**En résumé… 114**

**6. ÉCHANGES XML AVEC LES FOURNISSEURS** ..... 115
**Préparation de l'ajout des fonctionnalités XML 116**
Réalisation de la maquette de la page récapitulative de la commande 116
Encapsulation des fonctionnalités XML dans un objet métier 119
Création du composant XmlGenerator 120
Compilation du composant XmlGenerator 123
Intégration du composant dans la page récapitulative de la commande 124
**Génération des fichiers liés à la commande 125**
Génération du fichier XML à partir d'un objet DataSet 126
Génération d'une télécopie à l'aide d'une transformation XSL 127
Envoi du fichier XML par messagerie 130
**En résumé... 133**

**7. PERSONNALISER L'ERGONOMIE AVEC LES CONTRÔLES SERVEUR SPÉCIFIQUES** ..... 135
**Utilisation de contrôles serveur spécifiques 136**
Consultation des résultats de ventes par région avec VisualMap 136
Réalisation de la maquette de la page de consultation des ventes 136
Téléchargement et installation du contrôle serveur spécifique VisualMap 139
Intégration du contrôle VisualMap dans la page d'analyse des ventes 140
Paramètrage du contrôle VisualMap pour réaliser l'affichage des ventes par région 141
**Création d'un contrôle serveur spécifique 144**
Mécanisme de création d'un contrôle serveur spécifique 144
Personnaliser le contenu HTML produit par le contrôle avec la méthode Render 145
Architecture du contrôle Camembert 145
Création du contrôle Camembert 146
Implémentation de la génération de l'image au sein de la méthode Render 148
Éviter les problèmes de maintien en cache côté client avec la page LoadImage 149
Intégration du contrôle Camembert dans la page d'analyse des ventes 150
**En résumé... 152**

**8. Exposer et utiliser des services Web** ..... 153
**Implémentation d'un service Web de mise à jour des stocks 154**
Création d'un service Web avec ASP.NET 155
Utilisation de la classe de base WebService pour avoir accès à l'objet Session 156
Test du service Web de mise à jour des stocks 158
La page de test par défaut associée au service 158
Développement d'un proxy pour accéder au service Web 159
**Affichage de la météo en faisant appel à un service Web externe 161**
Recherche d'un service fournissant des informations météorologiques 161
Génération d'un proxy pour le service GlobalWeather 163
Implémentation du contrôle utilisateur Meteo 164
Réalisation de la maquette du contrôle Meteo 164
Implémentation du contrôle Meteo 165
**En résumé... 167**

**9. Sécurisation, optimisation et déploiement** ..... 169
**Sécurisation d'une application ASP.NET 170**
Redirection automatique vers une page d'authentification 170
Contrôle des autorisations en fonction de l'utilisateur 171
Les fichiers de configuration ASP.NET 172
Sécurisation d'un service Web 173
**Débogage et gestion des erreurs 175**
Analyser l'exécution d'une application 175
Accéder rapidement au déroulement détaillé avec l'option Trace 175
Examiner en détail le déroulement de l'exécution avec le débogueur .NET 177
Optimiser les performances de l'application grâce au maintien en cache 178
La gestion des exceptions 178
Gestion spécifique des erreurs 179
Spécifier une page d'erreur personnalisée 179
Intercepter l'ensemble des erreurs survenant dans une application 180
**Déploiement d'une application ASP.NET 181**
Déployer un assemblage dans le Global Assembly Cache 182
Gestion des différences de langue entre serveurs 183
**En résumé... 184**

**En conclusion...** ..... 185

**Annexes** ..... 187
**Fichiers de l'application 187**
**Structure de la base de données 188**
**Aide-mémoire C# et VB.NET 190**
**Liens utiles 192**

**Index** ..... 193

# L'étude de cas « Les Savons du Soleil »

# 1

ASP.NET

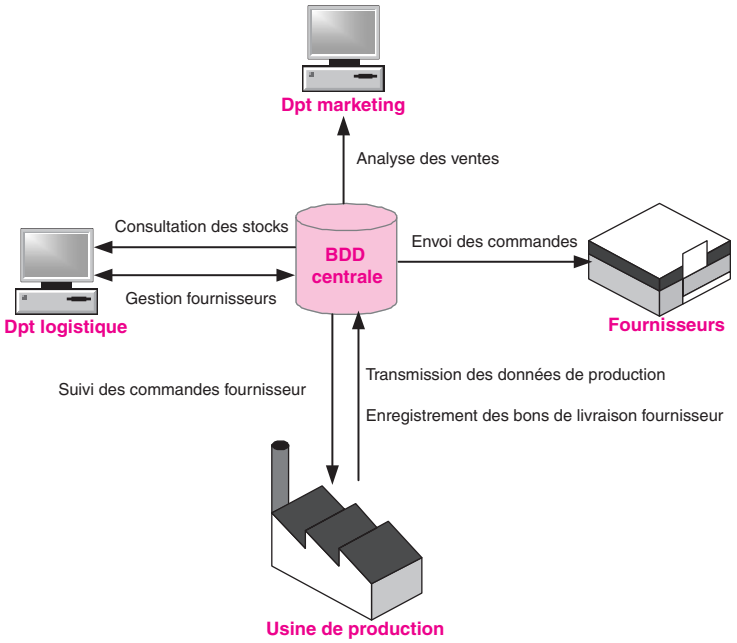
Base de données | Intranet | Échanges XML | Services Web | ASP.NET | MSDE | Web Matrix

## SOMMAIRE


- Présentation de la société
- Les problématiques actuelles
- Le projet de nouvelle infrastructure
- Justification des choix techniques


## MOTS-CLÉS

- Base de données centralisée
- Intranet
- Échanges XML
- Services Web
- ASP.NET
- MSDE
- Web Matrix




Dans ce chapitre, on présente en détail l'étude de cas qui servira de fil conducteur au reste de cet ouvrage : après avoir exposé les difficultés de circulation d'informations entre les différents sites de notre société exemple, on explique comment la mise en place une base de données centralisée régulièrement mise à jour depuis l'usine et dotée d'une interface de gestion adéquate peut améliorer la situation. Enfin, on détaille et on justifie les choix techniques retenus pour la réalisation de l'étude de cas : ASP.NET, MSDE et Web Matrix.

## Une PME géographiquement dispersée

La société des Savons du Soleil est une PME spécialisée dans la vente par correspondance de cosmétiques à base de produits naturels : savons au miel, shampoing au tilleul, bain douche à la lavande…

Les activités de la société sont réparties entre trois entités géographiquement séparées :

- *Marseille* : l'usine de production assure la fabrication et le stockage des produits, ainsi que le traitement des commandes (prise de commande, préparation et envoi des colis) ;
- *Lyon* : le département logistique assure le suivi des stocks et les commandes de matières premières auprès des fournisseurs ;
- *Paris* : le département marketing travaille sur les actions publicitaires, le développement de nouveaux produits et l'analyse des ventes.

## Des échanges d'informations fastidieux entre les sites

Les trois sites doivent échanger régulièrement des informations :

- le département logistique a besoin d'être renseigné sur l'évolution des stocks ;
- le département marketing doit analyser les chiffres de vente ;
- l'usine veut suivre les commandes passées par le département logistique.

Malheureusement, la remontée des informations depuis l'usine est complexe à gérer (figure 1–1), chaque unité possédant son propre système de gestion des données :

- l'usine de production utilise un logiciel de gestion commerciale dans lequel sont enregistrées toutes les commandes clients. À chaque fin de semaine, les informations relatives aux nouvelles commandes (coordonnées des clients, détail des produits commandés) sont extraites et transmises aux autres unités ;
- le département logistique suit l'évolution des stocks grâce à un tableur, dont le contenu est actualisé manuellement chaque semaine à partir des données fournies par l'usine ;
- le département marketing effectue ses analyses des ventes grâce à une base de données relationnelle, actualisée chaque semaine, elle aussi, à partir des informations de l'usine.

Quant à la gestion des commandes fournisseur – qui implique trois acteurs distincts – elle est, elle aussi, fastidieuse (figure 1–2) :

- le département logistique passe commande par télécopie auprès du fournisseur ;

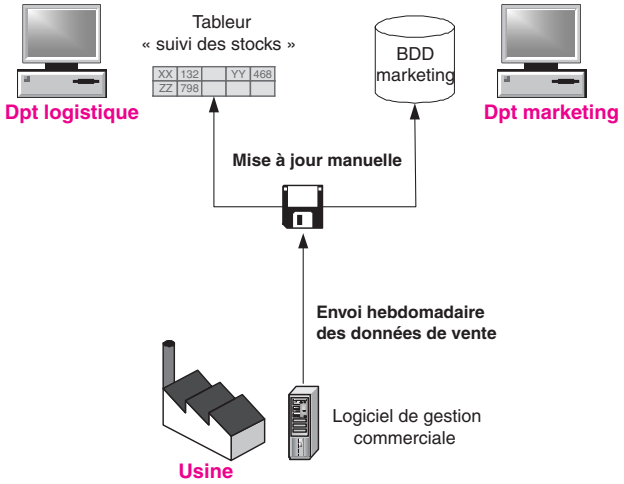



**Figure 1–1** Transfert des données commerciales (avant)

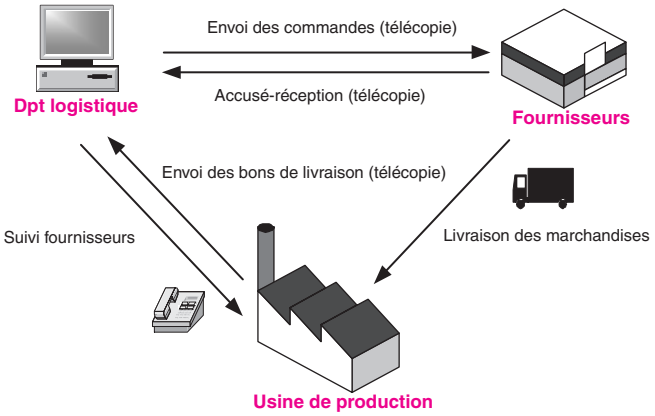



**Figure 1–2** Gestion des commandes fournisseur (avant)

- le fournisseur accuse réception de la commande, par télécopie également ;
- le bon de livraison remis par le fournisseur lors de la livraison de la marchandise est envoyé par télécopie, depuis l'usine, au département logistique.

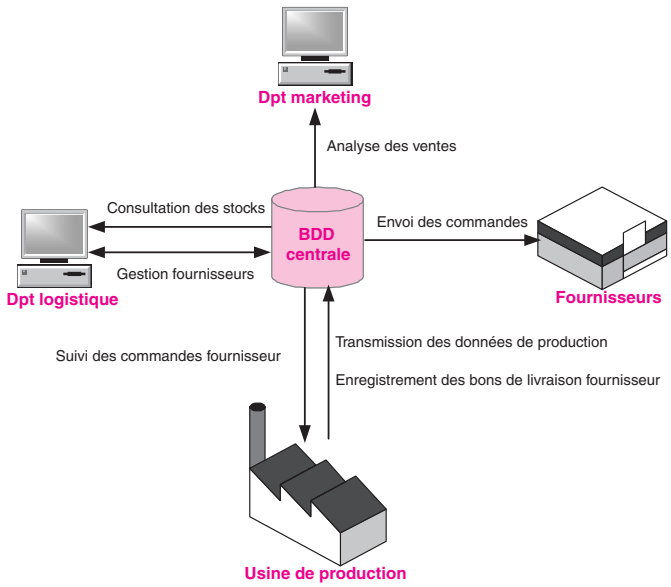
## Inconvénients liés à la situation actuelle

La situation actuelle présente un certain nombre d'inconvénients :

- les départements marketing et logistique n'ont pas accès aux informations « en temps réel », ce qui n'est pas dramatique pour le premier, mais nettement plus gênant pour le second ;
- les opérations de mise à jour des données à partir des informations envoyées par l'usine sont longues et fastidieuses ; de plus, elles contraignent le département marketing à employer à mi-temps un administrateur de base de données ;
- le département logistique ne dispose que de moyens limités d'analyse et souhaiterait disposer d'une base de données de suivi des stocks ;
- la gestion des commandes fournisseur est entièrement manuelle et pénible ;
- l'usine de production est contrainte de téléphoner au département logistique chaque fois qu'elle désire obtenir des informations sur les commandes fournisseur en cours.

En résumé, le fait d'avoir des sources d'information décentralisées est pénalisant pour l'activité de l'entreprise.

# Le projet de nouvelle infrastructure technique




Pour augmenter sa productivité au quotidien, la société des Savons du Soleil a donc décidé de mettre en place une nouvelle infrastructure technique :

- mise en place d'une base de données centralisée ;
- accès sécurisé à cette base pour toutes les unités via une interface Web permettant d'effectuer le suivi des stocks et des commandes fournisseur, ainsi que l'analyse des ventes ;
- mise à jour automatique et quotidienne de la base, à partir d'informations extraites des fichiers de gestion commerciale utilisés à l'usine ;
- nouveau mécanisme d'échange avec les fournisseurs, fondé sur XML.

Dans les sections qui suivent, nous détaillons les différentes composantes du projet :

- structure de la base de données ;
- interface de gestion de la base ;
- transferts de données automatisés (échanges fournisseur, mise à jour des données depuis l'usine).

**Structure de la base de données**

Pour simplifier la présentation de l'étude de cas – dont la base n'est pas le sujet principal – on se concentrera uniquement sur le modèle de données. Bien entendu, dans le contexte d'une application réelle, il faudrait aussi aborder d'autres sujets comme les règles d'intégrité référentielles (index, contraintes, déclencheurs), l'optimisation des performances, la gestion des transactions, la sécurité (authentification, rôles), la disponibilité de la base ou encore son administration.

## Mise en place d'une base de données centralisée

L'intérêt de mettre en place une base centrale est de disposer d'un référentiel de données unique partagé entre toutes les unités : on évitera ainsi les pertes de temps dues à la mise à jour manuelle des données de suivi des stocks et d'analyse des ventes, et on donnera à tous l'accès à une information à jour. Cette base, accessible via Internet, sera hébergée chez un prestataire externe.

Les données à stocker dans la base peuvent être réparties en deux sous-ensembles :

- données utilisées pour l'analyse des ventes et le suivi des stocks ;
- données utilisées pour la gestion des commandes fournisseur.

DANS UN CAS RÉEL

**Axes d'analyse plus nombreux**

Dans un cas réel, les axes d'analyse des ventes seraient beaucoup plus nombreux (par produits, par types de client, etc.). De même, le suivi de la logistique serait probablement plus complexe (gestion des stocks de produits intermédiaires...).

### Données nécessaires à l'analyse des ventes et au suivi des stocks

Pour éviter une trop grande complexité, on se limitera aux hypothèses suivantes :

- le département marketing souhaite pouvoir consulter la répartition des ventes par famille de produits ou par région, pour un mois donné ;
- le département logistique souhaite pouvoir suivre l'évolution des stocks de produits finis.

Pour effectuer ces opérations, il faut au minimum stocker dans la base :

- la liste des produits ;
- la liste des familles de produits ;

- la liste des mouvements de stocks (arrivée ou sortie du stock) ;
- la liste des ventes mensuelles par famille de produits et par région.

**Tableau 1–1** Données nécessaires à l'analyse des ventes et au suivi des stocks

| Entité | Caractéristiques, relations |
|---|---|
| Produit | • Désignation et description du produit<br>• Un produit est rattaché à une famille de produits |
| Famille de produits | • Nom de la famille de produits<br>• À une famille de produits sont rattachés de 1 à n produits |
| Mouvement de stocks | • Date, type de mouvement, produit concerné et quantité (positive ou négative)<br>• Un mouvement est rattaché à un produit |
| Vente | • Ventes par famille de produits et par région<br>• Une donnée de vente est rattachée à une famille de produits et à une région |

Le modèle physique correspondant, qui ne comporte aucune difficulté particulière, est présenté figure 1-3.

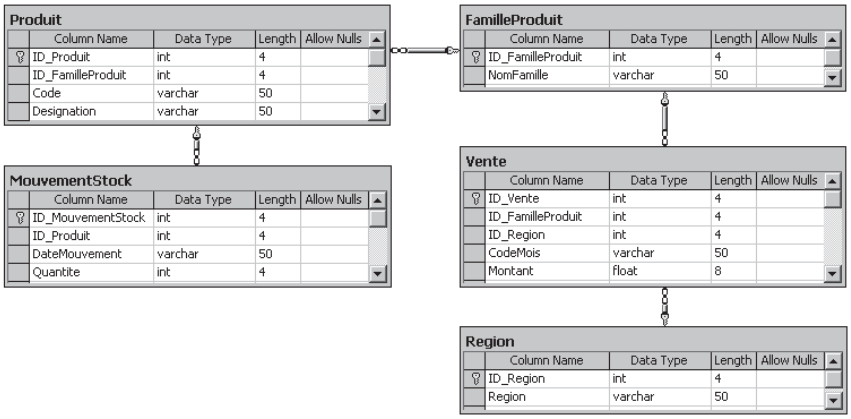



**Figure 1–3** Modèle physique de la base : analyse des ventes et suivi des stocks

**Notations**

Par convention, toutes les clés commencent par « ID_ » comme « IDentifiant ».

## Données nécessaires à la gestion des commandes fournisseur

Pour assurer le suivi des commandes fournisseur, le département logistique souhaite pouvoir garder trace des références fournisseur commandées (type et quantité), de la date de livraison prévue et du statut de la livraison (commande livrée ou non).

Par conséquent, il faut stocker dans la base :

- la liste des fournisseurs ;
- la liste des références fournisseur (produits proposés par ces fournisseurs) ;
- la liste des commandes passées auprès de ces fournisseurs (avec les lignes de commandes associées).

**Tableau 1–2** Données nécessaires au suivi des commandes fournisseur

| Entité | Caractéristiques, relations |
|---|---|
| Fournisseur | • Code fournisseur, nom et adresse<br>• À un fournisseur sont associées de 1 à n références fournisseur |
| Commande | • Date création, date livraison prévue, date livraison effective<br>• Une commande est rattachée à un fournisseur<br>• 1 à n lignes de commandes lui sont associées |
| Ligne de commande | • Référence fournisseur, quantité commandée, quantité livrée<br>• Rattachée à une commande |
| Référence fournisseur | • Référence et désignation du produit<br>• Associée à un fournisseur |

Le modèle physique correspondant est présenté figure 1-4.

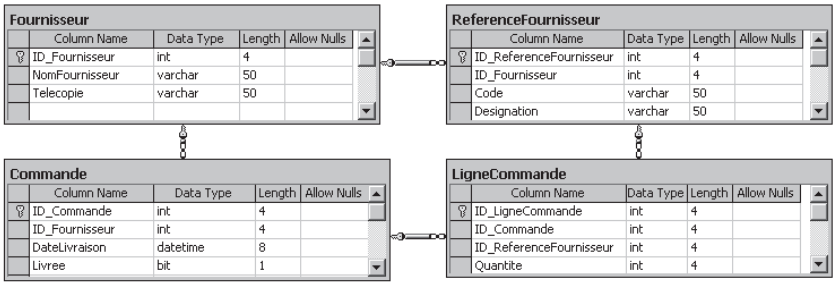



**Figure 1–4** Modèle physique de la base : gestion des commandes fournisseur

## Interface Web pour la consultation/mise à jour de la base

Les données de la base pourront être consultées et éventuellement mises à jour par les différentes unités de la société, en fonction de leurs besoins. Pour cela, les utilisateurs disposeront d'une interface de gestion permettant :

- le suivi des stocks ;
- la gestion de commandes fournisseur ;
- l'analyse des ventes.

Comme la base est hébergée sur un site externe, et que les utilisateurs se répartissent en trois endroits géographiquement distincts, ces opérations vont s'effectuer par l'intermédiaire d'une interface Web, dont l'accès devra être sécurisé (*intranet*).

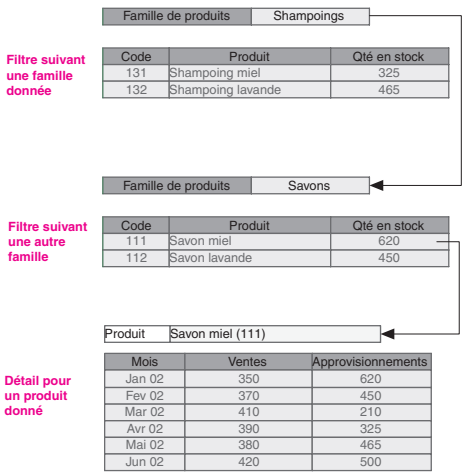



**Figure 1–5** Interface de suivi des stocks

## Module de suivi des stocks

Le module de suivi des stocks doit proposer :

- l'affichage de la liste des produits avec les quantités en stock correspondantes, filtrée par famille de produits ;
- l'accès à la fiche de détail d'un produit (historique de la variation du stock).

Toutes ces opérations sont présentées figure 1-5.

## Module de gestion des commandes fournisseur

Le module de gestion des commandes fournisseur doit permettre :

- la création d'une commande fournisseur ;
- l'affichage de la liste des commandes fournisseur ;
- l'enregistrement d'un bon de livraison fournisseur.

La création d'une commande fournisseur se déroule en plusieurs étapes (voir figure 1-6) :

1. Sélection du fournisseur.
2. Sélection des références, des quantités commandées et de la date de livraison souhaitée.
3. Enregistrement de la commande dans la base.
4. Transmission de la commande au fournisseur (sous forme de fichier XML).

La page de consultation des commandes fournisseur doit présenter une liste récapitulative des commandes : en sélectionnant une commande, on doit accéder au détail correspondant (références et quantité commandées, date de livraison prévue). Enfin, lors de la réception d'une commande à l'usine, il doit être possible de modifier le statut de la commande : « livrée » au lieu d' « en attente de livraison » (voir figure 1-7).

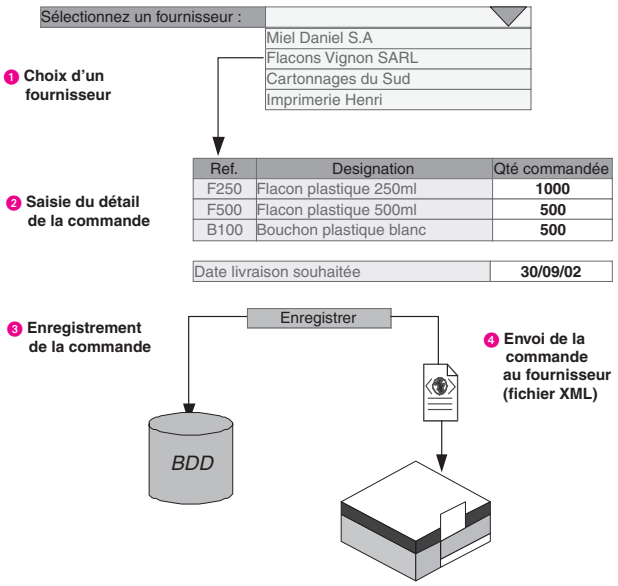



**Figure 1–6** Création d'une commande fournisseur

**Figure 1–7** Consultation des commandes fournisseur

## Module d'analyse des ventes

Ce module doit permettre la consultation du chiffre d'affaires réalisé par famille de produits ou par région sur une période donnée, avec visualisation des résultats sous forme graphique.

La cinématique correspondante est présentée figure 1-8.

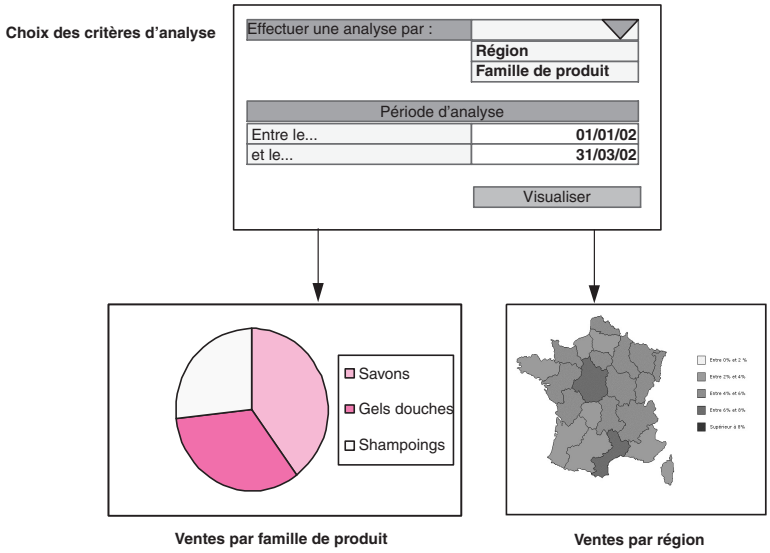



**Figure 1–8** Interface d'analyse des ventes

## Authentification

Cette base contient des données confidentielles qui ne doivent être consultées et modifiées que par les personnes qui y sont habilitées. Il est donc indispensable de prévoir un mécanisme d'authentification : les utilisateurs devront fournir un identifiant et un mot de passe valides pour pouvoir se connecter à l'application ;

**DANS UN CAS RÉEL Sécurisation**

Dans le cadre d'une application réelle, il faudrait passer en revue tous les aspects relatifs à la sécurité : protection du serveur Web par un pare-feu et un logiciel anti-virus, chiffrement des communications (SSL), authentification forte des utilisateurs, etc.

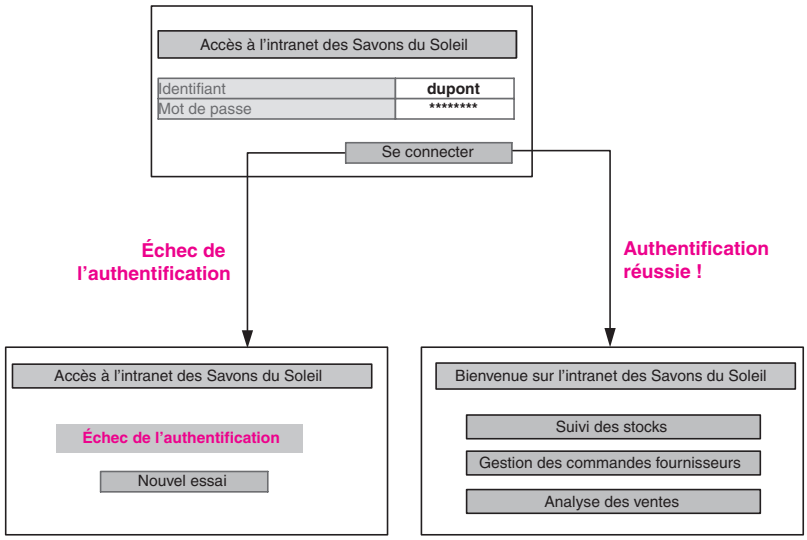



**Figure 1–9** Authentification des utilisateurs

de plus, l'accès aux différents modules dépendra des droits de l'utilisateur (gestion des autorisations). Le mécanisme requis pour l'authentification est présenté figure 1-9.

## Mise à jour automatique des données de la base

Le suivi des stocks et l'analyse des ventes n'auront de réel intérêt que si les données commerciales sont mises à jour régulièrement depuis l'usine : c'est pourquoi il est prévu de mettre en place un mécanisme de remontée d'informations depuis le logiciel de gestion commerciale vers la base centralisée.

Ce mécanisme reposera sur l'implémentation d'un service Web permettant la mise à jour de la base, qui sera appelé depuis l'usine de production (voir figure 1-10) ; dans un souci de simplification, on se limitera à la mise à jour des données de stocks.

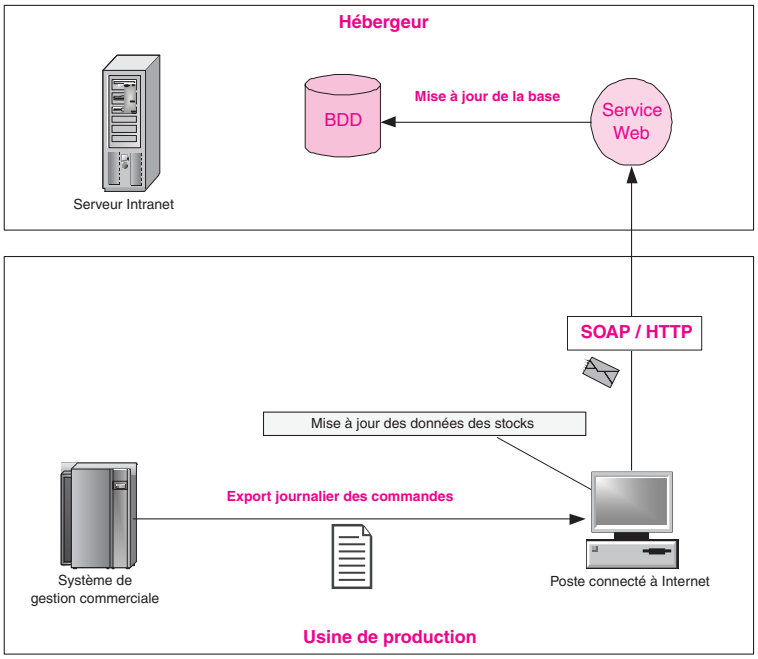



**Figure 1–10** Mise à jour des données de stock via un service Web

Le cahier des charges étant établi, nous allons maintenant choisir les outils techniques qui vont permettre sa réalisation.

**Utiliser un ERP ?**

Pour répondre de manière exhaustive aux besoins actuels et futurs de la société, il pourrait être envisagé de mettre en place un progiciel de gestion intégrée (ERP) : néanmoins, cette option représenterait un coût et un délai de mise en place jugés trop importants pour la société de notre étude de cas.

# Choix d'architecture technique : ASP.NET, MSDE et Web Matrix

Les utilisateurs étant situés dans des lieux distincts, équipés d'infrastructures techniques différentes, il est naturel de retenir une architecture de type intranet, reposant un serveur HTTP qui génère des pages dynamiques à partir d'informations contenues dans une base de données centralisée.

Ce type d'architecture nécessite de se déterminer sur les points suivants :

- choix de la technologie de développement Web ;
- choix de la base de données ;
- choix de l'environnement de développement.

## Choix de la technologie de développement Web

La technologie de développement constitue le moteur d'une application Web : c'est elle qui détermine le mécanisme de génération dynamique des pages Web ainsi les systèmes d'exploitation, bases de données, langages et outils de développement utilisables.

À l'heure actuelle, les choix possibles sont les suivants (par ordre alphabétique) :

| Technologie | Description succincte |
|---|---|
| ASP | Technologie de génération de sites Web dynamiques introduite par Microsoft en 1996, qui repose sur l'interprétation de scripts et l'activation d'objets ActiveX/COM côté serveur ; fonctionne uniquement sous Windows ; généralement utilisée avec Microsoft SQL Server. |
| ASP.NET | Nouvelle technologie de développement d'applications et de services Web, introduite par Microsoft en 2001, qui repose sur des pages compilées, des contrôles prédéfinis, un mécanisme de gestion événementielle et une infrastructure technique intégrant des fonctionnalités de sécurité, optimisation et gestion des erreurs ; fonctionne uniquement sous Windows 2000 et XP équipé de l'extension .NET Framework ; généralement utilisée avec Microsoft SQL Server. |
| ColdFusion | Nom désignant à la fois une technologie de type serveur d'application et un outil de développement, qui repose sur l'utilisation d'un langage spécifique (CFML : Coldfusion Markup Language) permettant d'activer des objets sur le serveur lors de l'interprétation d'une page Web ; fonctionne sous Unix, Linux, Windows, avec la majorité des serveurs HTTP et des bases de données ; historiquement développé par Allaire et récemment acquis par Macromedia (ColdFusion MX). |
| JSP | Technologie de génération de sites Web dynamiques développée par Sun, qui repose sur l'utilisation de pages compilées contenant du Java et l'activation d'objets serveurs (JavaBeans) ; fonctionne sur toutes les plates-formes équipées d'une machine virtuelle Java. |
| PHP | Langage de script open source créé en 1994, actuellement dans sa version 4, dont la syntaxe est inspirée de C / Perl et Java, permettant la génération de sites Web dynamiques ; fonctionne sur la majorité des plates-formes Unix, Linux et Windows et est compatible avec un grand nombre de bases de données ; couramment utilisé au sein du trio Linux/Apache/MySQL. |

Aujourd'hui, toutes ces technologies ont atteint un niveau de maturité assez avancé et, sur le papier, toutes permettraient de répondre aux besoins exprimés dans notre cahier des charges.

Néanmoins, ASP.NET se distingue de ses concurrents par son architecture novatrice et son nouveau modèle de programmation qui permet la réalisation d'applications Web avec les mêmes méthodes de conception (découpage en objets métier et composants réutilisables), le même degré de robustesse et de puissance (orientation objet, typage fort des variables, compilation) et le même degré d'organisation du code (séparation entre code et contenu graphique, organisation en gestionnaires d'événements) qu'une application client-serveur classique. Pour en savoir plus, nous allons détailler ces différentes caractéristiques de la technologie, en commençant par répondre à une question simple : qu'est-ce qu'ASP.NET ?

## Qu'est-ce qu'ASP.NET ?

En pratique, ASP.NET est une extension logicielle qui permet l'exécution d'applications Web (sites Web dynamiques, services Web). Disponible en téléchargement gratuit sur le site www.asp.net et destinée à être installée sur une machine équipée de Windows 2000/XP et d'un serveur HTTP, elle se compose des éléments techniques suivants :

- une extension du serveur HTTP, implémentée sous la forme de filtre ISAPI nommé `aspnet_filter.dll` (voir figure 1-11), qui permet de traiter spécifiquement les requêtes vers les pages comportant une extension particulière (notamment les fichiers `.aspx`, qui correspondent aux pages ASP.NET standards, voir figure 1-12) ;
- un processus principal nommé `aspnet_wp.exe` (pour : ASP.NET worker process) qui tourne en tâche de fond et traite les requêtes correspondant à des pages ASP.NET (figure 1-13), en s'appuyant lui-même sur trois éléments :

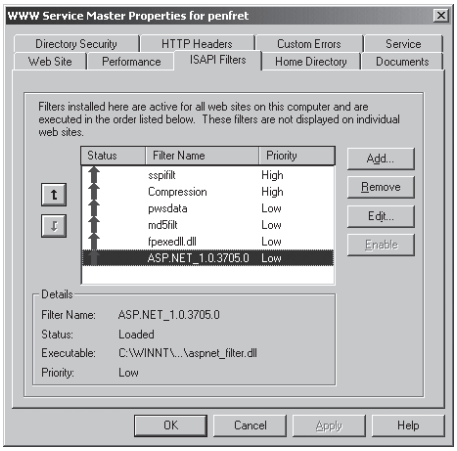



**Figure 1–11**
Le filtre ISAPI `aspnet_filter.dll`

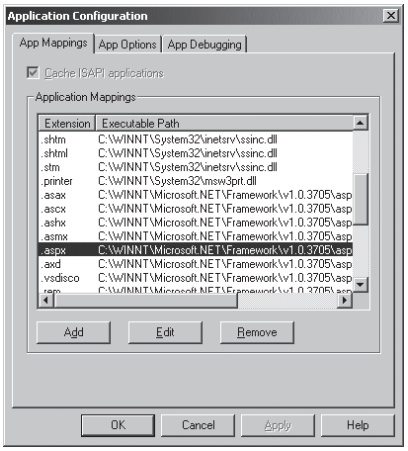



**Figure 1–12** Configuration des extensions pour ASP.NET

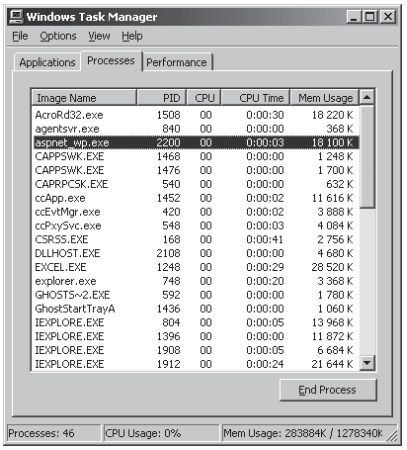



**Figure 1–13**
Le processus `aspnet_wp.exe`

**ASP.NET est indissociable du framework .NET**

De par son architecture, ASP.NET est indissociable du *framework .NET*, l'environnement d'exécution des applications .NET constitué du Common Language Runtime (CLR), de la bibliothèque de classes .NET et des compilateurs .NET

**Anecdote**

Le nom initial de la technologie ASP.NET était ASP+.

**Extension .aspx**

Les pages ASP.NET portent l'extension `.aspx` (pour : ASP eXtended).

**La compilation n'a lieu que lors de la première requête**

La compilation d'une page ASP.NET n'a lieu que lors du la première requête effectuée vers cette page ; l'assemblage produit est alors gardé en cache pour les requêtes successives (à moins que le fichier source ne soit modifié). Ceci garantit une performance optimale lors de l'exécution de l'application.

– les compilateurs .NET, qui permettent de compiler le code source des pages ASP.NET vers un format objet intermédiaire dit *Intermediate Language* (ou IL) ;
– une bibliothèque de classes utilitaires qui offre des fonctionnalités allant de l'accès aux données à la génération de documents XML, en passant par la réalisation de services Web ou de génération dynamique d'images, pour ne citer qu'une très brève étendue de ses possibilités ;
– une machine virtuelle nommée *Common Language Runtime* (CLR) prenant en charge l'exécution du code de la page (compilé en IL) et l'édition de liens avec les classes de la bibliothèque auxquelles le code de la page fait référence.

La figure 1-14 illustre le mécanisme d'appel d'une page ASP.NET :

- Lorsque le serveur HTTP reçoit une requête à destination d'une page portant l'extension `.aspx`, celle-ci est automatiquement traitée par le filtre ISAPI (`aspnet_filter.dll`) et transmise au processus principal ASP.NET (`aspnet_wp.exe`).
- La page demandée est alors compilée sous la forme d'un module objet en langage MSIL (Microsoft Intermediate Language) par le compilateur correspondant au langage utilisé dans la page (C#, VB.NET et JScript.NET sont les trois langages utilisables en standard) puis transmises à Common Language Runtime (CLR) qui réalise l'édition de lien avec les modules objet des classes utilisées de la bibliothèque .NET et la production d'un assemblage (composant binaire exécutable) dont l'exécution conduit à la production d'un contenu HTML, transmis en retour au navigateur.

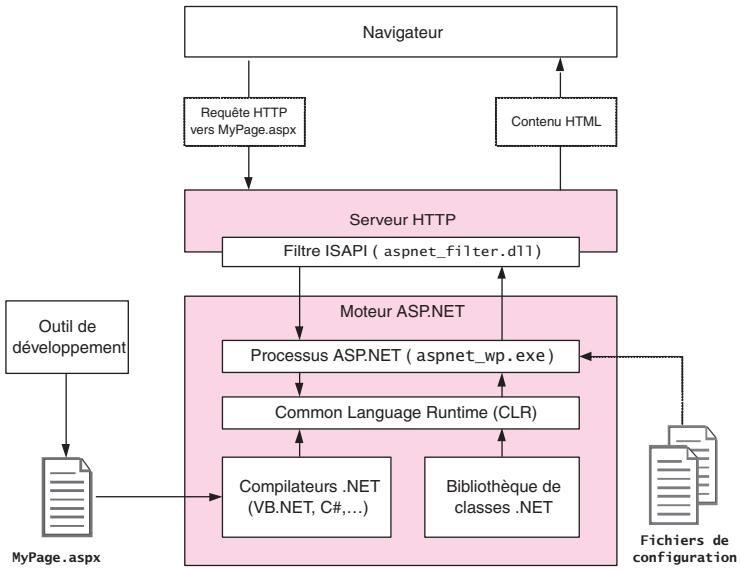



**Figure 1–14** Traitement d'une requête vers une page ASP.NET

Notons que la configuration du processus principal ASP.NET (langue, sécurité, gestion des erreurs) est entièrement paramétrable à l'aide de fichiers texte au format XML (`machine.config` et `web.config`), dont les changements sont détectés et pris en compte dynamiquement, sans nécessiter de redémarrage du serveur HTTP.

Nous avons décrit succinctement l'infrastructure d'exécution des pages ASP.NET (nous aurons l'occasion de présenter plus précisément le mécanisme de compilation et la notion d'assemblage dans le chapitre 3 et les possibilités de configuration dans le chapitre 9) : détaillons maintenant les nouvelles possibilités offertes au développeur pour la réalisation de pages Web.

**Parallèle avec Java**

Le langage MSIL est en quelque sorte équivalent au *byte code* Java, tandis que le Common Language Runtime est proche conceptuellement de la machine virtuelle Java. De plus, le mécanisme de compilation des pages à la volée est proche de celui implémenté par l'architecture JSP.

## Les nouveautés apportées par ASP.NET

Contrairement à une page ASP classique qui contient un entremêlement de contenu HTML et de script, les pages ASP.NET sont séparées en deux parties bien distinctes :

- la partie graphique constituée de balises et texte HTML (qui sera restitué tel quel), de contrôles serveur (éléments graphiques paramétrables qui seront, à l'exécution de la page, remplacés par un contenu HTML) et de contrôles utilisateur (fragments de pages HTML encapsulés au sein d'objets graphiques indépendants) ;
- la partie code qui spécifie le paramètrage des contrôles serveur et implémente la cinématique de la page en général, organisée au sein de gestionnaires d'événements et codée dans un des langages objet compatibles .NET : C# (proche de C++ et Java), VB.NET (proche de Visual Basic) ou JScript.NET (proche de JScript)

**Formulaires Web (Web Forms)**

Par analogie avec les formulaires Visual Basic, eux-mêmes constitués d'un assemblage de contrôle graphique dont on implémente la cinématique dans une partie code associée, les pages ASP.NET sont également appelées Formulaires Web (*Web Forms*).

**Langages utilisables dans une page ASP.NET**

À l'heure actuelle, le développeur a le choix entre trois langages par défaut pour développer une page ASP.NET :

- **VB.NET** : évolution majeure de Visual Basic auquel ont été principalement ajouté des fonctionnalités objet (constructeurs et destructeurs, surcharge de méthodes, héritage, etc.) ;
- **C#** : langage objet dont la syntaxe et les fonctionnalités sont proches de C++ et Java ;
- **JScript.NET** : évolution de JScript prenant en charge, entre autres, le typage des variables et les fonctionnalités objet (classe, héritage, etc.).

Ainsi, le développeur peut choisir le langage le moins éloigné de son champ de connaissance actuel et concentrer son attention sur les nouveautés de l'architecture ASP.NET plutôt que sur l'apprentissage d'un nouveau langage.

**Autres langages**

En théorie, une application .NET peut être développée avec tout langage conforme à la Common Language Specification (CLS) émise par Microsoft. Bien que le support d'une trentaine de langages ait été annoncé (dont Python, Perl, Eiffel, etc.), la mise en pratique peut s'avérer fastidieuse : nécessité de télécharger et d'installer un compilateur spécifique, absence d'exemples de code dans la documentation... C'est pourquoi les applications ASP.NET sont généralement implémentées avec VB.NET ou C#.

Les exemples de ce livre seront présentés dans les deux langages VB.NET et C#.

**Code disponible en ligne**

Dans certains cas, une version unique (VB.NET ou C#) sera proposée, dans un souci d'économie de place ; la version alternative sera alors consultable sur le site de l'étude de cas :

▸ www.savonsdusoleil.com

**Aide-mémoire**

Un aide-mémoire résumant les principales syntaxes des langages VB.NET et C# est fourni en annexe.

Cette architecture présente un certain nombre d'avantages :

- la séparation entre partie graphique et code augmente la lisibilité du code et facilite par conséquent sa maintenance ; d'autre part, elle permet, le cas échéant, de répartir plus facilement le travail entre des graphistes qui travaillent sur le contenu HTML et les développeurs qui se consacrent sur la logique applicative ;
- l'utilisation de langages orientés objet rend les développements plus robustes (typage fort, gestion des exceptions) et plus simples à réutiliser (héritage, polymorphisme) ;
- l'organisation du code en gestionnaires d'événements (« exécuter tel code lorsque tel événement survient »), rendu possible par un mécanisme capable de réaliser des allers-retours (*round-trip*) entre le navigateur et le serveur HTTP en conservant les valeurs contenues dans les contrôles, facilite l'implémentation de pages interactives (par exemple : mise à jour des produits disponibles lorsque l'utilisateur sélectionne une famille de produits).

Nous développerons tous ces sujets dans le chapitre 3, où nous étudierons l'architecture d'une page ASP.NET (séparation entre contenu graphique et code, technique du *code behind*, notion de contrôle serveur) ; la notion de gestionnaire d'événement sera, quant à elle, décrite en détail à la fin du chapitre 4.

**Compatibilité ascendante avec ASP**

Toute application ASP peut s'exécuter dans un environnement ASP.NET.

Par ailleurs, les pages ASP.NET bénéficient de la richesse de la bibliothèque .NET qui comporte plusieurs milliers de classes utilitaires permettant entre autres :

- la consultation et la mise à jour de données contenues dans une base, notamment à l'aide de contrôles serveur liés aux données comme `DataGrid`, `DataList` et `Repeater` (voir chapitres 4 et 5) ;
- la création et la manipulation de documents XML ainsi que la réalisation de transformations XSL (voir chapitre 6) ;
- l'envoi de fichiers par messagerie (voir chapitre 6) ;
- la génération dynamique d'images (voir chapitre 7) ;
- l'implémentation et l'appel de services Web (voir chapitre 8).

## Choix de la base de données

Dans une application ASP.NET comme dans la grande majorité des sites Web dynamiques, la base de données est une composante fondamentale de l'architecture. L'offre logicielle en la matière est vaste ; parmi les produits les plus connus disponibles sous Windows, citons : Microsoft SQL Server, Oracle, PostgreSQL, Informix, 4D, Microsoft Access…

**Pourquoi pas Microsoft Access ?**

Il est vrai que Microsoft Access a toujours constitué une alternative facile à utiliser et peu coûteuse à SQL Server. Néanmoins, le fait que le moteur Jet ne soit pas géré nativement par .NET, conjugué à la gratuité de MSDE plaide en défaveur du choix de cet outil pour notre étude de cas.

À l'heure actuelle, la bibliothèque .NET propose les types d'accès suivants :

- accès natif à Microsoft SQL Server (et à sa version allégée : MSDE) ;
- accès natif à Oracle (fournisseur disponible en téléchargement séparé) ;
- pilote OLE-DB ;
- pilote ODBC (fournisseur disponible en téléchargement séparé).

Pour notre étude de cas, nous nous proposons d'utiliser la base Microsoft SQL Server Desktop Engine (MSDE) en raison de sa gratuité.

## MSDE : une version allégée et gratuite de SQL Server 2000

Microsoft SQL Server Desktop Engine (MSDE) est une version allégée et gratuite de Microsoft SQL Server, doté d'un support complet de Transact SQL (y compris la gestion des transactions, des déclencheurs et de la sécurité). En revanche, elle n'offre qu'un support partiel de la réplication, ne dispose pas d'outils graphiques d'administration (comme SQL Enterprise Manager ou Analyseur de requête) et son usage reste limité à un nombre réduit d'utilisateurs simultanés.

Cette base constitue néanmoins une solution bien adaptée pour un poste de développement, du fait de sa faible consommation de ressources, d'une part, mais également de sa compatibilité totale avec SQL Server 2000, qui facilite un éventuel déploiement vers un serveur de production doté de ce SGBD. L'installation de MSDE sera décrite dans le chapitre suivant.

**À propos de la similarité des moteurs SQL**

La version précédente du moteur MSDE (1.0) était différente de SQL Server 7.0 ; désormais, MSDE 2000 et SQL Server 2000 sont dotés du même moteur.

**Si vous disposez de SQL Server**

Bien entendu, les lecteurs qui le souhaitent pourront utiliser SQL Server au lieu de MSDE : les exemples donnés dans cet ouvrage seront parfaitement applicables à l'un comme à l'autre.

## Choix de l'environnement de développement

Dans l'absolu, il est possible de développer une application ASP.NET avec un simple éditeur de texte : fastidieuse, cette approche a néanmoins l'avantage indéniable de permettre au développeur de maîtriser tout le code qu'il écrit, et certains en sont partisans !

À l'opposé, il existe des environnements de développement qui se proposent d'automatiser les tâches laborieuses à l'aide d'assistants, de faciliter le confort de travail du développeur (coloration syntaxique, complétion automatique du code saisi, compilateurs intégrés, aide en ligne), citons :

- **Visual Studio.NET** : très complet, il gère de nombreux types d'applications (ASP.NET, WinForms, MFC, ATL), plusieurs langages (C, C++, VB.NET, C#), est doté de fonctionnalités puissantes (grand nombre d'assistants, aide contextuelle, technologie Intellisense pour compléter le code) et est adapté au travail en groupe (gestion de projets complexes, intégration avec des outils de contrôles de source) ;
- **Web Matrix** : environnement léger dédié au développement d'applications ASP.NET ; moins riche en fonctionnalités que Visual Studio.NET (pas d'aide contextuelle, pas de complétion automatique du code, pas de notion de projet), il n'en est pas moins doté d'un grand nombre d'assistants (modèles de pages, ajout de contrôles, gestion de la base de données) et présente l'indéniable avantage d'être gratuit ;
- **Delphi 7 Studio** : la nouvelle version du célèbre environnement de développement édité par Borland fournit désormais un support pour le développement d'application ASP.NET avec le langage Delphi.

Dans le cadre d'une grosse équipe de développement travaillant sur des projets importants, l'emploi de Visual Studio.NET ou de Delphi serait probablement nécessaire ; pour notre étude de cas, limitée à une application ASP.NET, réalisée a priori par un développeur seul, Web Matrix répond amplement aux besoins.

**Utiliser quand même VS.NET ou Delphi**

Bien entendu, les lecteurs qui le souhaitent pourront tout de même tirer parti de cet ouvrage en utilisant Visual Studio.NET ou Delphi au lieu de Web Matrix : en effet, l'accent sera plus porté sur la technologie ASP.NET proprement dite que sur les outils ou langages utilisés pour la mettre en œuvre.

**Web Matrix : version 0.5 (en anglais)**

À l'heure actuelle, Web Matrix n'est disponible qu'en version 0.5 (Technology Preview), en anglais. Si quelques fonctionnalités font encore défaut, cette version est globalement satisfaisante en terme de stabilité (et les quelques commandes de menu en anglais ne devraient pas dérouter le lecteur). Notons que dans sa version finale, le produit sera toujours gratuit.

- www.asp.net/webmatrix

**Code in-line vs code behind**

Une des différences fondamentales entre Web Matrix et Visual Studio.NET réside dans la gestion de la séparation entre code et contenu HTML : Web Matrix place le code et le contenu HTML d'une page dans le même fichier, tout en offrant une séparation visuelle entre les deux à l'aide d'onglets (*code in-line*), tandis que Visual Studio.NET place le code associé à une page dans un fichier séparé du contenu HTML (*code behind*). Nous aurons l'occasion de détailler ces deux techniques dans le chapitre 3.

### Web Matrix : un environnement de développement efficace et gratuit

Web Matrix est un environnement léger dédié au développement ASP.NET. Disponible en téléchargement gratuit sur www.asp.net/webmatrix (son installation sera détaillée dans le chapitre suivant), il démontre la volonté de Microsoft de fédérer une communauté de développeurs autour d'ASP.NET : outil gratuit, nombreux exemples de sources, accès à des forums depuis l'interface.

Le tableau ci-dessous résume les principales différences entre Visual Studio.Net et Web Matrix :

| | Visual Studio.NET | Web Matrix |
|---|---|---|
| Assistants graphiques | X | X |
| Gestion base de données | X | X |
| Complétion du code (Intellisense) | X | |
| Client FTP intégré | | X |
| Serveur HTTP intégré | | X |
| Gestion de projets | X | |
| Contenu du code source | X | |
| Compilation | Intégrée | Séparée (ligne de commande) |
| Séparation HTML/code | Fichiers séparés (code behind) | Même fichier (code in-line) |
| Version actuelle | 7.0 | 0.5 |
| Coût licence | Variable selon le type de licence | Gratuit |

Bien qu'il soit moins riche en fonctionnalités que Visual Studio.NET ou Delphi, Web Matrix s'avère tout à fait adapté au développement ASP.NET : les nombreux assistants et modèles, le client FTP intégré et le serveur HTTP personnel font de cet outil gratuit un compagnon fidèle et efficace.

## En résumé

Dans ce premier chapitre, nous avons posé les bases de notre étude de cas : après avoir présenté les activités des « Savons du Soleil » et les problématiques auxquelles est confrontée cette société, nous avons décrit en détail le projet d'évolution du système d'information.

Puis, nous avons proposé une vue d'ensemble de la technologie ASP.NET et justifié le choix de la base de données MSDE et de l'environnement de développement Web Matrix pour la réalisation de notre étude de cas.

Dans le chapitre suivant, nous allons décrire l'installation de ces outils sur le poste de travail.

# Installation des outils et création de la base

# 2

ASP.NET

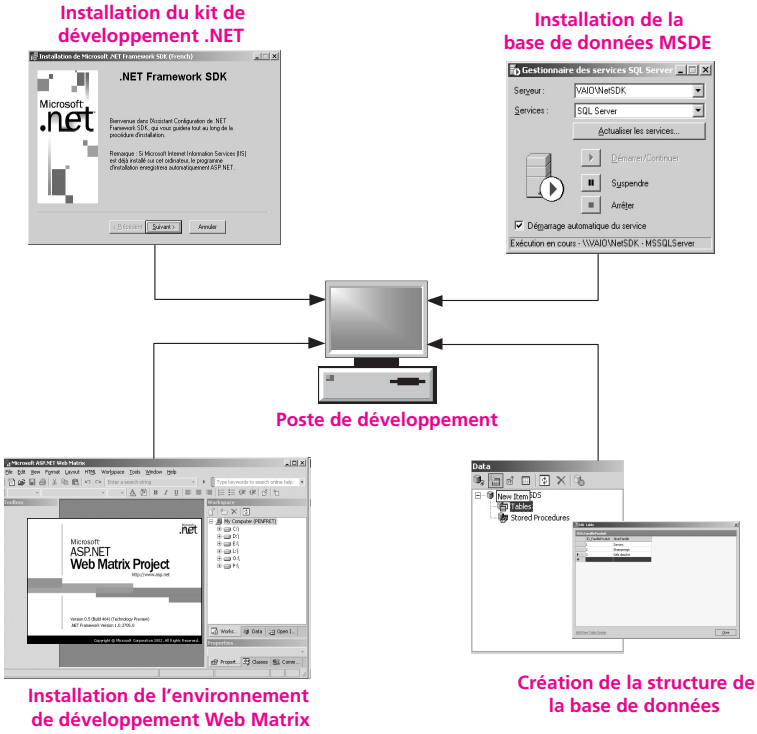
.NET Framework | ASP.NET | MSDE | Web Matrix | Scripts SQL

## SOMMAIRE


- Installation des outils
  - Kit de développement .NET
  - Base de données MSDE
  - Environnement Web Matrix
- Création de la structure de la base


## MOTS-CLÉS

- .NET Framework
- ASP.NET
- MSDE
- Web Matrix
- Scripts SQL




La réalisation de notre application ASP.NET nécessite l'installation, sur le poste de développement, de plusieurs outils : le kit de développement .NET, le moteur de base de données MSDE et, enfin, un environnement de développement léger et gratuit, Web Matrix. Dans ce chapitre, nous montrons comment installer ces outils et les utiliser pour mettre en place la structure de la base de données.

# Installation des outils de développement

## Obtention du kit de développement .NET

Le kit de développement .NET (ou .NET SDK en anglais) contient l'ensemble des outils nécessaires au développement d'applications .NET :

- le Common Language Runtime (CLR) ;
- la bibliothèque de classes .NET;
- les compilateurs des langages .NET (VB.NET, C# et JScript.NET) ;
- le moteur de bases de données MSDE (Microsoft Desktop Data Engine) ;
- la documentation complète ;
- de nombreux exemples de code.

Ce kit de développement est téléchargeable gratuitement depuis le site www.asp.net à l'adresse www.asp.net/download.aspx (130 Mo, nécessite une connexion rapide) :

1. Choisissez l'option Download .NET Framework SDK : vous êtes alors redirigé vers la page « .NET SDK » du site de Microsoft (figure 2-1).
2. Choisissez la langue de votre choix (le kit est disponible en français).
3. Choisissez de télécharger le kit en une seule fois (130 Mo) ou en plusieurs fois (13 fichiers de 10 Mo).

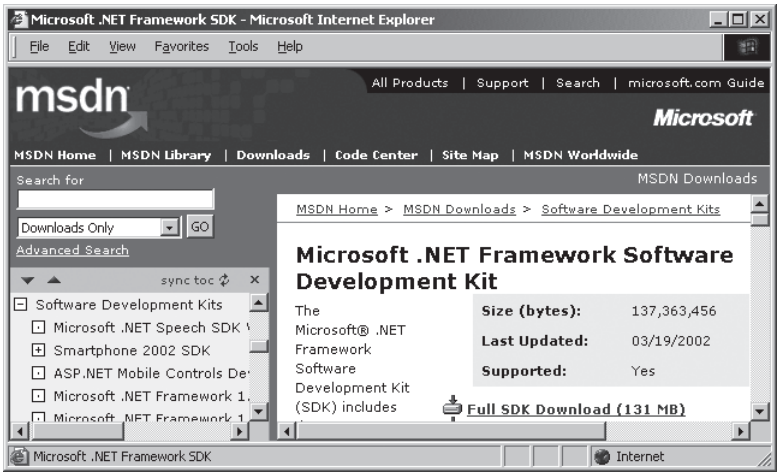



**Figure 2–1** Télécharger le kit de développement .NET

Il faut avouer que la taille de ce téléchargement est conséquente mais le jeu en vaut la chandelle !

### .NET SDK

Dans la documentation en anglais, ce kit de développement est appelé « .NET SDK » pour *.NET Software Development Kit*.

▸ www.asp.net/download.aspx

### ALTERNATIVE Obtention du .NET SDK

Le .NET SDK est aussi livré aux abonnés MSDN (CD « Microsoft .NET Framework SDK ») et fourni en standard avec Visual Studio.NET sur le CD-Rom Windows Component Update.

### Ne peut-on obtenir ASP.NET seul ?

Même si vous n'êtes intéressé que par la réalisation d'applications ASP.NET, vous devrez télécharger l'ensemble du kit de développement .NET !

### .NET Redistributable vs .NET SDK

Le kit *.NET Redistributable* ne contient que le minimum vital pour pouvoir faire fonctionner une application .NET, à savoir le Common Language Runtime, les bibliothèques de classes et les compilateurs – il est généralement installé sur les serveurs de production. Le *.NET SDK* contient en plus la documentation, des exemples de code et la base de données MSDE.

### ASTUCE Optimiser le téléchargement

Étant donné la taille des fichiers à télécharger, il est recommandé d'utiliser un logiciel FTP capable de reprendre un téléchargement interrompu ; par exemple, Fresh Download de :

▸ http://www.freshdevices.com

## Prérequis à l'installation du kit de développement .NET

Il est indispensable de disposer d'une machine équipée d'un système d'exploitation compatible avec ASP.NET :

- Windows 2000 – toutes versions (Service Pack 2 recommandé).
- Windows XP Professional.

D'autre part, il est recommandé d'installer préalablement le serveur Web IIS 5.0 ainsi que la version 2.7 de MDAC (figure 2-2).

**ATTENTION ASP.NET ne fonctionne pas avec NT4**

Bien que le framework .NET puisse être installé sur Windows NT4 (Service Pack 6a requis), la technologie ASP.NET ne fonctionne qu'avec Windows 2000 et XP.

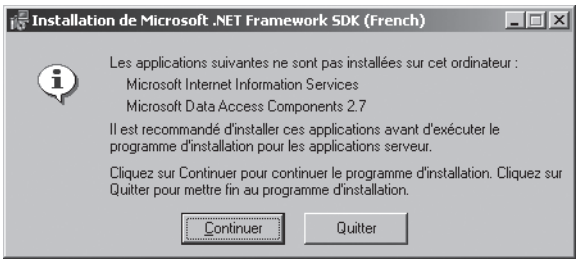



**Figure 2–2** Installation d'IIS et MDAC recommandée

## Installation du serveur Web Internet Information Server (IIS)

L'installation d'Internet Information Server (IIS), serveur HTTP fourni en standard dans Windows, est recommandée dans la mesure où elle permet de profiter des exemples fournis avec le kit de développement NET.

Pour installer Internet Information Server :

1. Choisissez Ajout/Suppression de programmes dans le panneau de configuration.
2. Choisissez Ajouter/Supprimer des composants Windows.
3. Cochez la case Internet Information Server et cliquez sur OK. Le CD-Rom d'installation de Windows sera requis au cours de l'installation.

**ALTERNATIVES Serveur HTTP de Web Matrix ou Apache**

Si vous ne souhaitez pas installer IIS sur votre machine, vous pouvez utiliser le serveur HTTP léger fourni avec Web Matrix pour visualiser les pages ASP.NET. D'autre part, notez qu'il existe une version d'Apache compatible avec ASP.NET.

▸ www.covalent.com

## Installation du kit de pilotes de bases de données MDAC

Microsoft Data Access Components (MDAC) est un kit logiciel gratuit contenant tous les pilotes de bases de données nécessaires pour Windows. L'installation de la dernière version (2.7 et au-delà) est indispensable pour utiliser la bibliothèque d'accès aux données ADO.NET.

Pour installer MDAC 2.7 :

1. Téléchargez MDAC 2.7 sur www.microsoft.com/data (environ 5 Mo).
2. Exécutez le programme d'installation (aucune option à choisir).

**VERSION Quelle est votre version de MDAC ?**

Pour connaître le numéro de version du kit MDAC installé sur votre poste, téléchargez l'outil « Component Checker » à l'adresse :

▸ www.microsoft.com/data

**ATTENTION Mise à jour de MDAC**

Dans certains cas, l'installation de MDAC 2.7 peut poser des problèmes de compatibilité avec des applications antérieures.

▸ www.microsoft.com/data/MDAC27Info/en/readmerefresh.htm

## Désinstallation des versions beta antérieures

Enfin, avant d'installer le kit de développement .NET, il est nécessaire d'avoir préalablement désinstallé les éventuelles versions beta de .NET présentes sur la machine.

**Versions du framework .NET**

À ce jour, trois versions du framework .NET se sont succédées, qui sont incompatibles entre elles :

- la *beta1* (1.0.2204),
- la *beta2* (1.0.2914)
- la version 1.0 (1.0.3705).

Curieusement, toutes trois sont numérotées 1.0.xxxx, bien que les deux premières soient des versions beta !

Il est prévu que la prochaine version du framework (1.1) soit livrée en standard dans .NET Server 2003 (et compatible avec la version 1.0).

▸ www.microsoft.com/windows.netserver

Ces installations préliminaires étant effectuées, votre poste est maintenant prêt pour l'installation du kit de développement .NET.

## Installation du kit de développement

**Utilisateurs de Visual Studio.NET**

Si vous installez le .NET Framework à partir de Visual Studio, insérez le disque 1 et laissez-vous guider.

Une fois l'exécutable d'installation téléchargé, la procédure est très simple:

1. Lancez `setup.exe`.
2. Un écran de choix apparaît : gardez les options par défaut (figure 2-3).
3. Cliquez sur Suivant et patientez quelques dizaines de secondes… c'est fini !

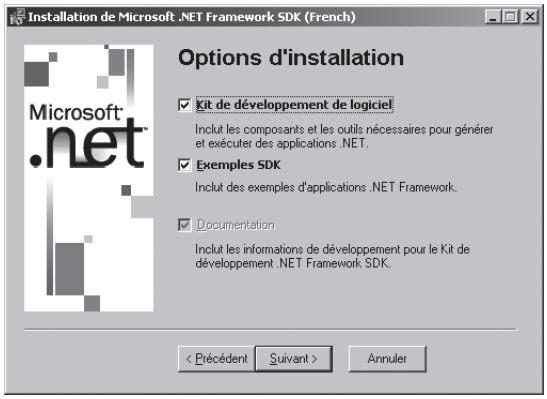



**Figure 2–3** Options d'installation du kit de développement .NET

L'installation du framework .NET est maintenant terminée :

- la bibliothèque de classes et les programmes nécessaires au fonctionnement de l'architecture .NET (CLR, compilateurs, processus ASP.NET, fichiers de

configuration) ont été installés sur votre poste de travail, dans `\WINNT\Microsoft.NET\Framework\v.1.0.3705` ;

- la documentation et les fichiers exemples ont été copiés sur le disque dur (une étape de configuration supplémentaire devra être effectuée après l'installation de la base de données) dans le répertoire d'installation `\Program Files\Microsoft.NET\FrameworkSDK` ;
- des raccourcis ont été ajoutés dans le menu **Démarrer** (figure 2-4).

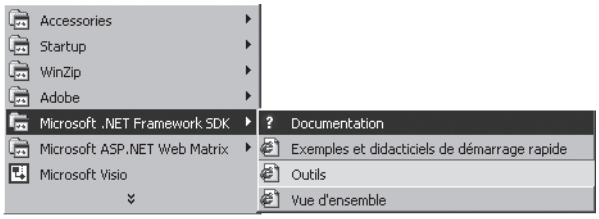



**Figure 2–4** Les raccourcis ajoutés dans le menu Démarrer

Passons maintenant à l'étape suivante : la mise en place de la base de données MSDE.

## Installation de Microsoft SQL Server Desktop Engine (MSDE)

L'installation de MSDE est très simple :

1. Localisez l'exécutable d'installation `instmsde.exe` situé dans le répertoire `\Program Files\Microsoft.NET\Framework SDK\Samples\setup\msde`.
2. Lancez l'installation (elle est rapide et aucune option n'est à choisir) : un écran de progression apparaît (figure 2-5).
3. L'installation est terminée lorsque cet écran disparaît : redémarrez alors la machine.

Au redémarrage, la seule trace visible de l'installation de MSDE est l'icône du gestionnaire des services SQL Server, qui permet de faire apparaître la boîte de dialogue présentée figure 2-6 : ne changez rien aux options si vous souhaitez que MSDE démarre automatiquement lors du chargement de Windows.

MSDE ne comprend aucune interface graphique de gestion de la base (contrairement à SQL Server qui est fourni avec la console d'administration Entreprise Manager et divers utilitaires dont un analyseur de requêtes, des outils de configuration réseau).

Heureusement, il est possible de configurer MSDE via l'environnement de développement Web Matrix, que nous allons maintenant installer, juste après avoir configuré les exemples du kit de développement .NET.

**Installation du service pack le plus récent**

Après l'installation, il est recommandé de télécharger et d'appliquer le service pack le plus récent relatif au framework .NET.

▸ http://msdn.microsoft.com/netframework/

**Utilisateurs de SQL Server**

Si SQL Server est déjà installé sur votre poste, il n'est évidemment pas nécessaire d'installer MSDE.

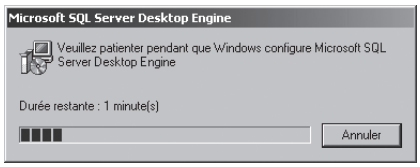



**Figure 2–5** Installation de MSDE

Curieusement, aucun message n'annonce la fin de l'installation de MSDE !

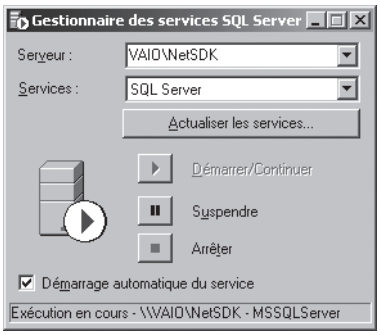



**Figure 2–6** Gestionnaire des services SQL Server

## Installation des exemples du kit de développement .NET

Maintenant que la base MSDE est installée, il est possible d'installer les exemples livrés avec le kit de développement .NET :

1. Localisez l'exécutable d'installation `ConfigSamples.exe` situé dans le répertoire `\Program Files\Microsoft.NET\Framework SDK\Samples\setup`.
2. Lancez l'installation (aucune option n'est à choisir) : l'utilitaire crée un certain nombre de répertoires virtuels dans IIS et des bases de données exemples dans MSDE.
3. Un message apparaît pour signaler la fin de l'installation (figure 2-7).

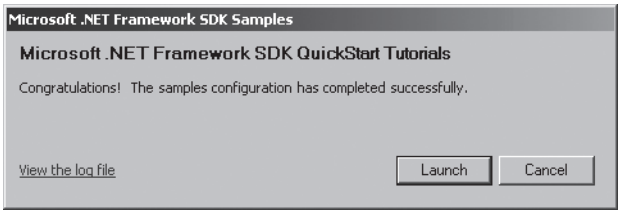



**Figure 2–7** Installation des exemples livrés avec .NET

On peut alors cliquer sur Launch pour faire apparaître les exemples, qui illustrent les principales fonctionnalités d'ASP.NET et auxquels vous pouvez d'ores et déjà jeter un coup d'œil !

## Téléchargement et installation de Web Matrix

**Internet Explorer 5.5 requis**

L'installation de Web Matrix requiert Internet Explorer 5.5 ou supérieur. Téléchargement gratuit à l'adresse :

- www.microsoft.com/windows/ie

L'installation de Web Matrix est très rapide :

1. Téléchargez l'exécutable d'installation sur http://www.asp.net/webmatrix (gratuit).
2. Lancez l'exécutable d'installation : un écran apparaît (figure 2-8).
3. Cliquez sur Next (vous pouvez changer le répertoire de destination).
4. Patientez quelques dizaines de secondes… c'est terminé !

L'installation a ajouté un groupe ASP.NET Web Matrix dans le menu Programmes : nous allons tout de suite utiliser Web Matrix pour créer la structure de la base de données.

# Création de la structure de la base de données

Le lecteur trouvera en annexe tous les détails relatifs à la structure de la base à mettre en place (schéma des tables, listes des procédures stockées…) et pourra télécharger directement un script de création complet de la base à l'adresse :

- www.savonsdusoleil.com

Plusieurs options sont possibles pour créer la structure de la base de données : utiliser Web Matrix, utiliser la console SQL Server Enterprise Manager, utiliser le mode « projets de données » de Microsoft Access, ou enfin avoir recours à des scripts SQL.

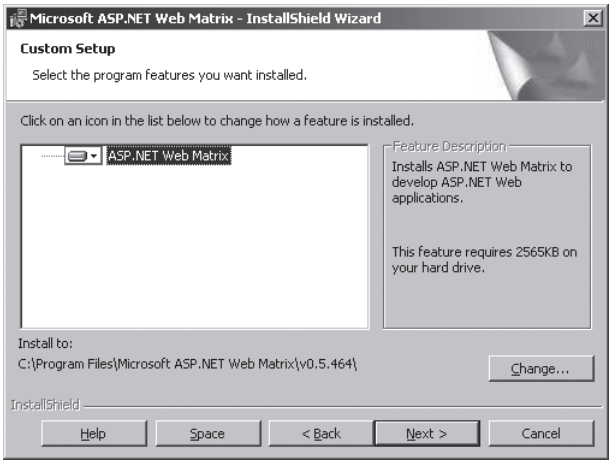



**Figure 2–8** Installation de Web Matrix

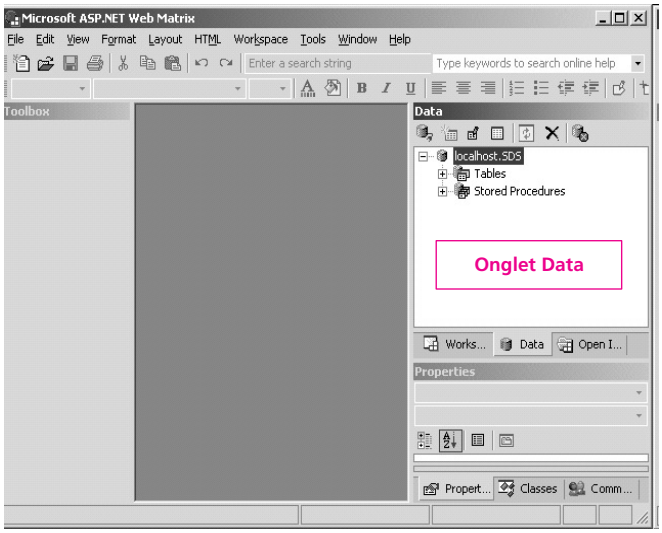



**Figure 2–9** L'onglet Data de Web Matrix

Dans cette section, nous ne faisons que décrire le fonctionnement général de ces outils, en détaillant plus particulièrement les fonctionnalités proposées par Web Matrix.

## Création de la structure de la base avec Web Matrix

L'environnement Web Matrix dispose d'une interface graphique permettant d'effectuer des opérations simples sur une base de données SQL Server ou MSDE :

- créer une base de données (ou se connecter à une base existante) ;
- créer ou modifier des tables (structure et données) ;
- créer ou modifier des procédures stockées.

Toutes les opérations de manipulation de données sont effectuées depuis l'onglet Data de la fenêtre Workspace, situé en haut à droite de l'environnement de développement (figure 2-9).

**IMPORTANT Nom du serveur de données**

Lors de l'installation de MSDE, une instance nommée (*named instance*) de SQL Server est créée : pour accéder au serveur, il faut utiliser l'adresse `(local)\NetSDK`.

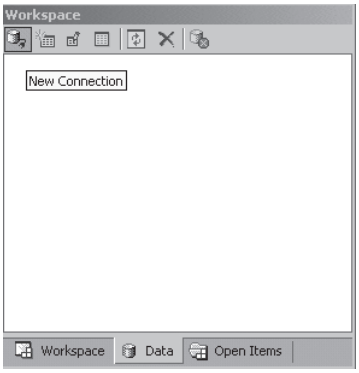
### Création d'une nouvelle base

Pour créer une nouvelle base ou se connecter à une base existante avec Web Matrix :

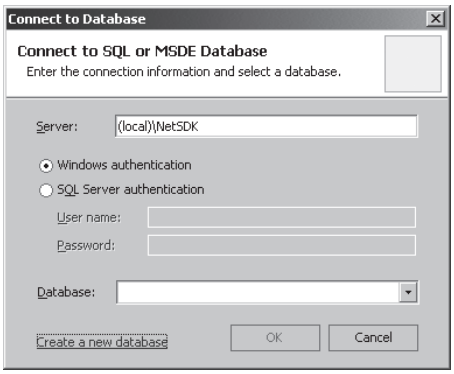
1. Cliquez sur l'icône New Connection dans l'onglet Data ❶. Une fenêtre de connexion à un serveur de données apparaît ❷.
2. Spécifiez le nom du serveur et le mode d'authentification souhaité.
3. Cliquez sur le lien Create a new database et saisissez un nom de base ❸.

**Localisation des fichiers de la base**

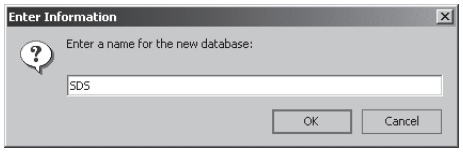
Les fichiers de la base sont situés, par défaut, dans le répertoire `\Program Files\Microsoft SQL Server\MSSQL$NetSDK\Data`.




1 Créer une nouvelle base avec Web Matrix




2 Connexion à un serveur de données




3 Création d'une nouvelle base

### Modifier la structure d'une table existante

Un lien *Edit/View Table Design* situé en bas de la fenêtre de saisie de données dans une table permet de consulter ou modifier la structure de cette table. Malheureusement, il n'est possible de modifier la structure que si la table est vide !

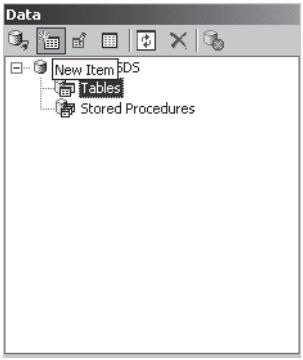
## Création de tables

Une fois connecté à une base de données, voici comment créer une table avec Web Matrix :

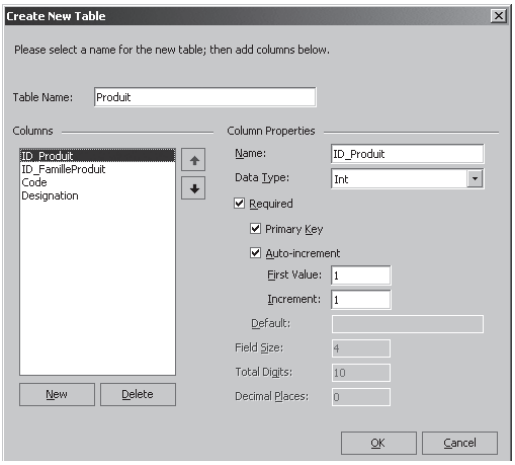
1. Sélectionnez l'élément Tables dans la fenêtre Data.
2. Cliquez sur l'icône New Item 4. Un écran permettant de spécifier les caractéristiques de la table apparaît 5.
3. Spécifiez le nom de la table ainsi que le nom et les caractéristiques de chaque colonne (nom, type, attributs : champ obligatoire, clé primaire, numéro automatique, etc.).
4. Cliquez sur OK pour enregistrer les modifications.

Pour mettre à jour le contenu d'une table existante :

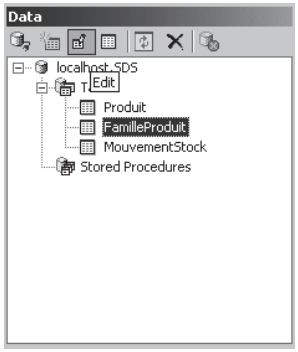
1. Sélectionnez la table souhaitée dans la fenêtre Data et cliquez sur l'icône Edit 6.
2. Saisissez les données dans la fenêtre d'édition qui apparaît 7.



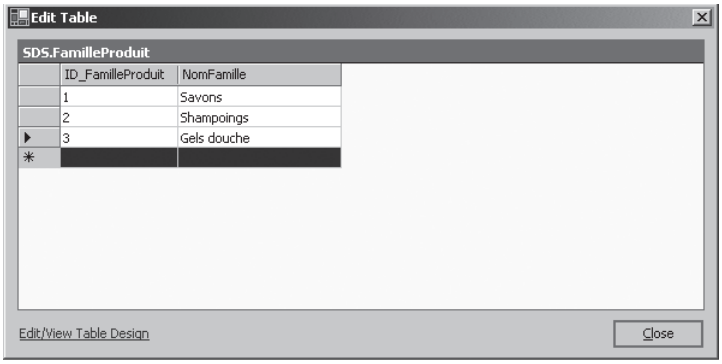

4 Création d'une nouvelle table




5 Définition de la structure d'une table




6 Édition des données d'une table




7 Saisie de données dans une table

## Création de procédures stockées

Web Matrix permet de créer des procédures stockées, en saisissant directement le code SQL dans une fenêtre de saisie :

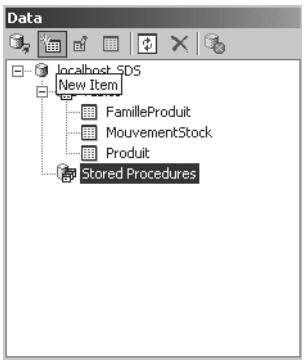
1. Sélectionnez l'élément Stored Procedures dans la fenêtre Data.
2. Cliquez sur l'icône New Item 8 : une fenêtre destinée à l'édition du code SQL de la procédure apparaît 9.
3. Saisissez le code de la procédure.
4. Cliquez sur OK pour enregistrer les modifications.

**WEB MATRIX Vues SQL non gérées !**

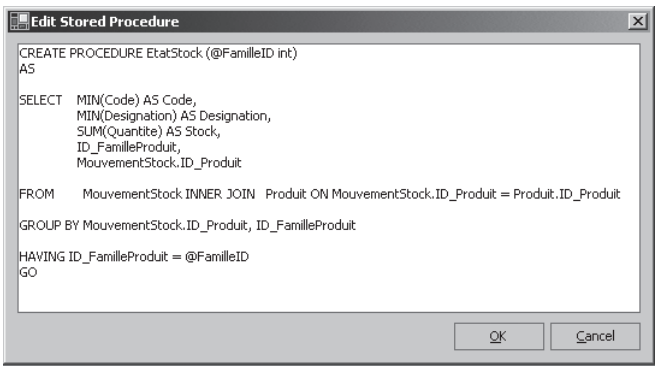
À l'heure actuelle, Web Matrix ne permet malheureusement pas de créer des vues SQL : nous nous limiterons donc à l'utilisation de procédures stockées dans notre étude de cas.

## Conclusion sur l'interface de gestion de données de Web Matrix

Comme vous avez pu le constater, les possibilités de Web Matrix en matière de gestion des données sont relativement réduites : il n'est en particulier pas possible de créer des relations, des vues, ni de modifier la structure d'une table lorsqu'elle est remplie, encore moins de gérer les permissions, les index ou les déclencheurs… Néanmoins, Web Matrix permet aux utilisateurs ne disposant pas de Microsoft Access ni de SQL Server Entreprise Manager d'effectuer les opérations de base sans avoir recours à l'utilisation de scripts.




8 Création d'une nouvelle procédure stockée




9 Saisie du code d'une procédure stockée

### Création de relations entre tables

Dans une base correctement structurée, on établit des relations entre les tables pour s'assurer de l'intégrité des données. Malheureusement, Web Matrix ne propose *pas* d'interface graphique pour la création de relations entre tables : à moins de disposer d'une version complète de SQL Server ou de Microsoft Access, le seul moyen de créer des relations est donc d'utiliser des scripts SQL en mode ligne de commande !

Pour cela, on utilise l'outil `osql.exe`, fourni en standard avec MSDE et installé par défaut dans le répertoire `\Program Files\Microsoft SQL Server\80\Tools\Binn`, qui permet d'exécuter des scripts SQL sur une base de données (voir figure 2-10).

L'outil `osql` se démarre en mode ligne de commande DOS avec la syntaxe suivante :

```
osql -S<NomServeur> -d<NomBase> -E
```

où :

- `<NomServeur>` désigne le nom du serveur de données ;
- `<NomBase>` désigne le nom de la base de données ;
- `-E` indique qu'il faut la sécurité intégrée de Windows pour authentifier la connexion.

Voici ensuite la syntaxe SQL à utiliser pour la définition d'une relation :

```
> ALTER TABLE <TableEtrangere>
  ADD CONSTRAINT <NomContrainte>
  FOREIGN KEY (<CleEtrangere>)
  REFERENCES <TablePrimaire> (<ClePrimaire>)
> GO
```

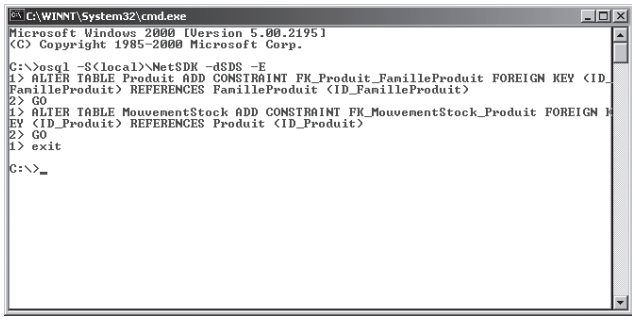


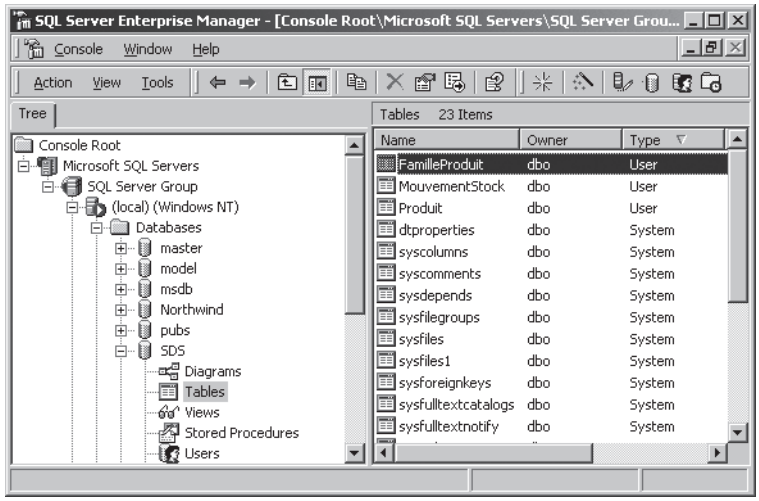

**Figure 2–10** Créer les relations en mode ligne de commande

## Alternative 1 - Utilisation de la console SQL Server Enterprise Manager

Si vous possédez une édition de SQL Server, vous pouvez effectuer toutes les opérations de création de la structure de la base à l'aide de la console SQL Server Enterprise Manager ❶.

On ne décrira pas ici l'ensemble des possibilités de cette interface très complète, pour laquelle Microsoft fournit une aide en ligne à l'adresse :

▸ www.microsoft.com/sql




❶ L'interface SQL Server Enterprise Manager

## Alternative 2 - Utilisation de Microsoft Access

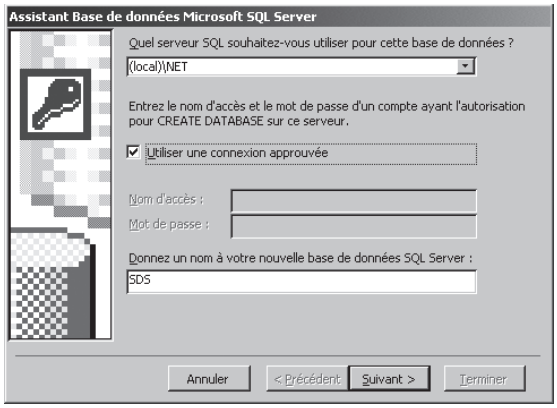
Microsoft Access dispose d'une fonctionnalité peu connue mais néanmoins très riche en possibilités : les *projets de données*, qui permettent de gérer des bases de données MSDE ou SQL Server en utilisant Access comme interface graphique de gestion (et non pas également en tant que moteur de base de données, comme c'est généralement le cas).

Pour créer une base SQL avec Microsoft Access :

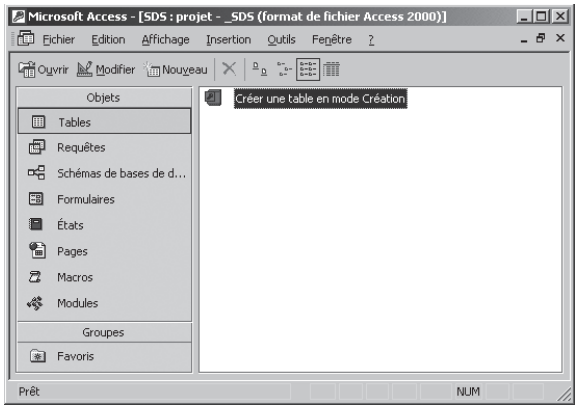
1. Créez un nouveau projet de données (fichier `.adp`).
2. Paramétrez les informations relatives à la base ❷.
3. Créez la structure de la base avec l'interface graphique ❸.

Un article complet sur ce mode d'utilisation d'Access est disponible sur le site de Microsoft :

- http://office.microsoft.com/france/assistance/2002/articles/WaysToWorkWithSQLData.aspx




❷ Création d'une base SQL avec Microsoft Access




❸ Création d'une base SQL avec Microsoft Access

## Alternative 3 - Utilisation de scripts SQL

Enfin, il est possible d'utiliser un script SQL pour créer la structure de la base et peupler des tables, en utilisant l'exécutable en ligne de commande `osql`.

La syntaxe de la ligne de commande est la suivante :

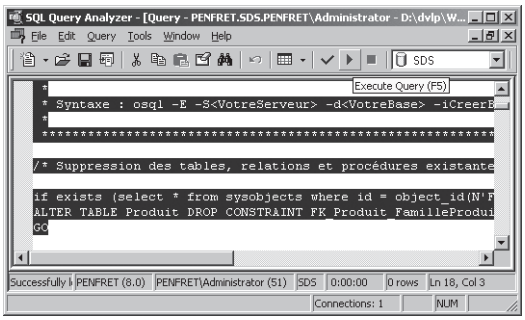
```
osql -S<NomServeur> -d<NomBase> -i<NomFichierScript> -E
```

où :

- `<NomServeur>` désigne le nom du serveur de données ;
- `<NomBase>` désigne le nom de la base de données ;
- `<NomFichierScript>` désigne le nom du fichier de script à appliquer ;
- `-E` indique qu'il faut la sécurité intégrée de Windows pour authentifier la connexion.

**ALTERNATIVE Utiliser l'analyseur de requête SQL Server**

Pour ceux qui possèdent SQL Server, il suffit d'ouvrir le fichier de script avec l'analyseur de requêtes SQL, de sélectionner tout le texte et de cliquer sur le bouton Execute Query




Exécution d'un script avec l'analyseur de requête SQL

**RAPPEL Mise en place de la base pour l'étude de cas**

Avant de passer au chapitre suivant, le lecteur est invité à mettre en place la structure de la base de données de l'étude de cas à l'aide des scripts SQL disponibles sur le site :

- www.savonsdusoleil.com

## En résumé…

Après avoir installé le kit de développement .NET, la base de données MSDE et l'outil de développement Web Matrix, nous avons décrit les différentes alternatives possibles pour la mise en place de la base de données : l'interface de gestion des données de Web Matrix, le mode « projets de données » de Microsoft Access, la console SQL Server Entreprise Manager et l'utilisation de scripts SQL.

Dans le chapitre suivant, nous allons voir concrètement à quoi ressemble une page ASP.NET, comment est réalisée la séparation entre code et présentation et ce que sont les contrôles serveur et les contrôles utilisateur.

# Architecture d'une page ASP.NET

# 3

ASP.NET

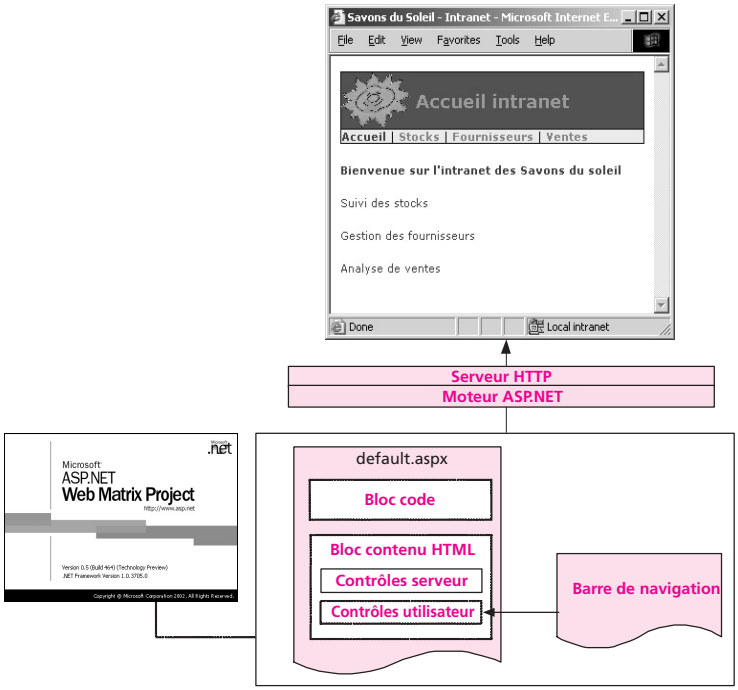
Contrôle serveur | contrôle utilisateur | Code behind | Web Matrix | attribut CssClass

## SOMMAIRE

- Modèle de programmation ASP.NET
- Notion de contrôle serveur
- Séparation contenu graphique/ code
- Notion de contrôle utilisateur
- Réalisation d'une barre de navigation

## MOTS-CLÉS

- Contrôle serveur
- Contrôle utilisateur
- Code behind
- Web Matrix
- Directive <%@Page ...%>
- Directive <%@Register...%>
- Directive <%@Control...%>
- Attribut CssClass




Dans ce chapitre, nous présentons l'architecture générale d'une page ASP.NET : quels sont les concepts principaux du nouveau modèle de programmation qui leur est associé, comment s'effectue en pratique la séparation entre le code et le contenu graphique, comment le développement devient plus simple et plus productif grâce à l'utilisation de *contrôles serveur* et *contrôles utilisateur*. Puis nous mettons en pratique ces connaissances pour réaliser la barre de navigation de notre étude de cas.

## La page Web vue comme une interface classique

Le principe fondamental d'ASP.NET est de considérer une page Web non plus comme un document HTML mais comme une interface graphique classique d'application client-serveur.

Tout l'intérêt d'ASP.NET est de permettre au développeur d'implémenter une page Web comme un assemblage de *contrôles* graphiques (listes, boutons…) réagissant à des *événements* (changement de sélection dans une liste, clic sur un bouton…), tout en assurant la traduction de cette vision conceptuelle en HTML standard.

Prenons un exemple inspiré de notre étude de cas, l'interface de consultation de la liste des produits des Savons du Soleil, classés par famille de produits :

- Dans le cadre d'une application client-serveur classique, cette interface aurait été implémentée avec un langage objet, sous la forme d'une boîte de dialogue contenant des contrôles – une liste déroulante affichant la liste des familles et une grille de données affichant la liste des produits – avec une cinématique associée à des événements : remplissage des contrôles lors du chargement de l'interface, mise à jour de la liste des produits dans le cas d'un changement de sélection de famille.
- Dans le cadre d'une application Web classique, cette interface aurait été implémentée sous la forme d'une page Web dynamique, contenant du HTML mêlé à du langage de script, sans possibilité de gérer de manière transparente un changement de sélection de famille de produits.

Dans le passé, il fallait donc choisir entre une interface classique, riche en fonctionnalités implémentées avec un langage puissant, mais imposant un déploiement sur le poste client, et une interface Web (plus limitée en possibilités graphiques) n'intégrant pas ou peu de gestion événementielle et implémentée avec un langage de script, même si cela éliminait le problème du déploiement.

Désormais, la technologie ASP.NET permet de profiter des avantages de l'approche classique tout en bénéficiant de l'architecture Web : implémentation des pages Web comme un assemblage de contrôles, utilisation de langages évolués, gestion événementielle, performance liée à la compilation (figure 3-1).

Rassurez-vous, rien de magique dans tout cela, si ce n'est une restructuration complète du modèle de programmation de pages Web dynamiques, initialement introduit par ASP et désormais principalement articulé autour des mécanismes suivants :

- les contrôles serveur ;
- le séparation entre code et contenu graphique ;
- les contrôles utilisateur ;
- la gestion des événements « côté serveur ».

**Contrôle**

Un contrôle est un élément graphique paramétrable ; l'utilisation de contrôles rend le code plus modulaire et accroît la productivité du développement.

**Programmation événementielle**

Modèle de programmation consistant à organiser le code d'une application sous la forme de gestionnaires d'événements, c'est-à-dire de traitements exécutés lorsque tel ou tel événement survient. Technique couramment utilisée lors du développement d'interfaces graphiques.

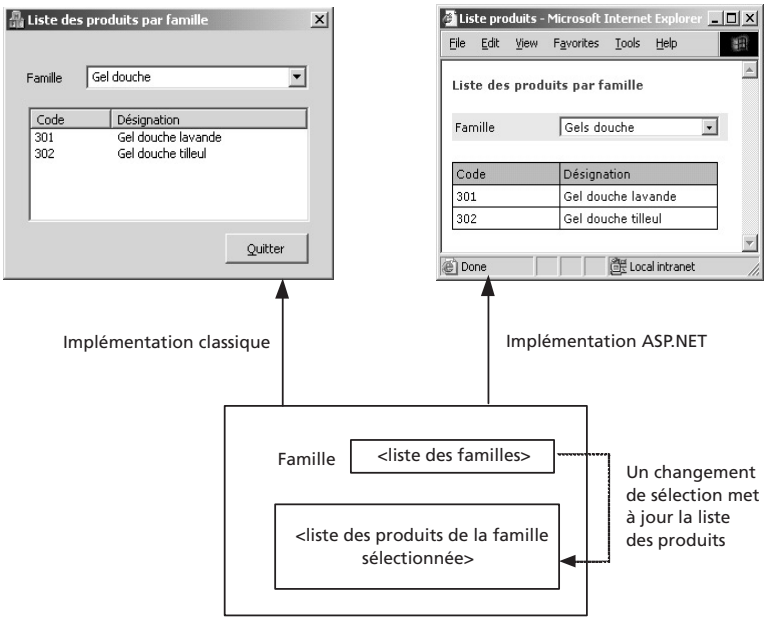



**Figure 3–1** Deux implémentations d'une même vision conceptuelle

Nous allons détailler et mettre en pratique les trois premiers mécanismes dans la suite de ce chapitre, tandis que nous traiterons la gestion des événements dans le chapitre suivant.

**À retenir**

ASP.NET permet d'appréhender une page Web comme une interface graphique d'application client-serveur lors de la phase de développement, tout en produisant des pages HTML standards lors de la phase d'exécution.

## Un contenu HTML mieux organisé grâce aux contrôles serveur

Comme pour une interface client classique, le développement d'une page ASP.NET se déroule en deux phases distinctes :

- la *réalisation de la maquette*, qui consiste à positionner les éléments graphiques sur la page en isolant, en particulier, les composants paramétrables ;
- l'*implémentation de la cinématique*, qui consiste à spécifier les actions à réaliser en fonction des événements qui surviennent.

Poursuivons avec notre exemple de page Web qui affiche la liste des produits ; cette page contient deux éléments paramétrables :

- une liste déroulante contenant les familles de produits, à remplir lors du chargement de la page ;
- un tableau de données contenant les produits de la famille sélectionnée, à remplir lors du chargement de la page et à actualiser lors de chaque changement de sélection dans la liste des familles.

APARTÉ

**Un mode de programmation familier ?**

Les développeurs d'applications client-serveur classiques (Visual Basic, Visual C++, Delphi…) sont familiers de ce modèle de programmation : ils découvriront dans ce chapitre comment ASP.NET l'applique dans le développement Web.
Les développeurs plus habitués à la réalisation de pages Web fondées sur des langages de scripts (VBScript, JScript…) découvriront la puissance de cette nouvelle approche, notamment en termes de productivité et de structuration du code.

L'implémentation correspondante dans une page ASP.NET consistera en un document HTML classique, dans lequel on insérera des balises spéciales pour chacun des composants paramétrables :

- une balise `<asp:DropDownList>` pour la liste des familles ;
- une balise `<asp:DataGrid>` pour la liste des produits.

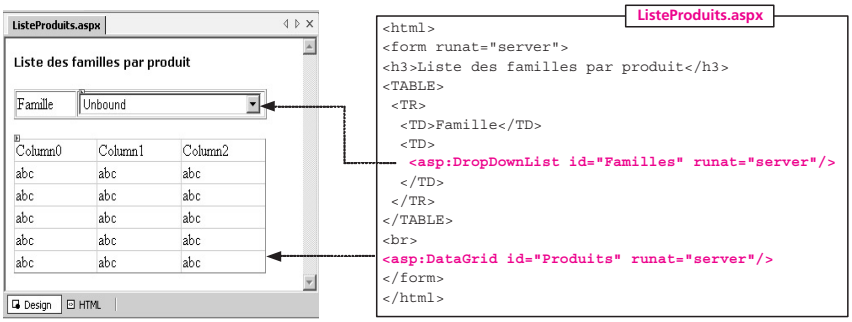



**Figure 3–2** Conception de la maquette d'une page ASP.NET

Dans le vocabulaire ASP.NET, ces balises correspondent à des *contrôles serveur* : elles seront remplacées par de l'HTML généré dynamiquement lors de l'exécution de la page (figure 3-2).

### Contrôle serveur

Composant graphique paramétrable, que l'on peut insérer dans une page ASP.NET et qui encapsule la génération de HTML. Les contrôles serveur participent à la séparation entre code et contenu graphique et accroissent la productivité du développement en facilitant la réutilisation.

### Plus d'informations sur les contrôles serveur

Les contrôles serveur sont des éléments graphiques déclarés, positionnés dans la partie HTML de la page et accessibles sous forme d'objets dans la partie code.

La déclaration d'un contrôle serveur s'effectue de la manière suivante :

```
<asp:TypeControle runat = "server"
 id="NomDuControle"></asp:TypeControle>
```

On peut aussi employer la syntaxe abrégée :

```
<asp:TypeControle runat = "server"
 id="NomDuControle" />
```

Chaque contrôle serveur propose des propriétés permettant de contrôler son apparence et son comportement (taille, couleur, format, source de données liées, etc.). Lors de l'exécution de la page, les contrôles serveur génèrent un contenu HTML, en fonction de leur paramétrage et du navigateur utilisé par le client (adaptation du HTML généré en fonction des possibilités du navigateur).

Lors d'un « aller-retour » de la page suite à un événement donné, les contrôles serveur conservent leur valeur (grâce au mécanisme du **`VIEWSTATE`**). Enfin, il est possible d'associer des gestionnaires d'événements à un contrôle serveur.

Rassemblés dans l'espace de nommage **`System.Web.UI`**, les contrôles serveur fournis avec le framework .NET peuvent être répartis en plusieurs groupes :

- Les **contrôles HTML** : correspondant aux éléments HTML standards (boutons, zones de texte, liens…) , ils offrent l'avantage d'exposer leurs attributs sous forme de propriétés. Ils sont employés lorsqu'on souhaite contrôler l'apparence des éléments HTML depuis le code.
- Les **contrôles Web** : équivalents aux contrôles HTML standards, ils proposent des propriétés personnalisées qui aident au développement. Ils sont très fréquemment employés dans les pages ASP.NET.
- Les **contrôles de validation** : ils facilitent les opérations de validation de formulaires en vérifiant le contenu des champs (diverses règles de validation possibles) et en indiquant les champs erronés. De plus, le code de validation généré est adapté au navigateur du client.
- Les **contrôles liés à des données** : très utilisés dans les applications Web qui manipulent des données, ils permettent la présentation de listes de données et l'édition d'enregistrements.
- Les **contrôles complexes** : on regroupe dans cette catégorie les contrôles évolués tels que le calendrier ou le contrôle `AdRotator`, qui permet d'afficher des publicités en alternance.
- Les **contrôles mobiles** : ensemble de contrôles générant du contenu adapté à des terminaux mobiles (téléphone, Pocket PC…).

Enfin, notons qu'il est possible d'implémenter des contrôles serveur spécifiques ; il existe d'ailleurs un marché important de contrôles serveur proposés par des vendeurs indépendants.

Par exemple, le contrôle `DataGrid` (grille de données) sera remplacé à l'exécution par un tableau HTML :

```
<table>
   <tr>
   <td >Code</td><td>Désignation</td>
   </tr><tr>
      <td>101</td><td>Savon miel</td>
   </tr><tr>
      <td>102</td><td>Savon miel & amandes</td>
   </tr><tr>
      <td>103</td><td>Savon lavande</td>
   </tr>
</table>
```

Nous aurons l'occasion, au cours de cet ouvrage, de détailler les différents types de contrôles serveur ainsi que leurs possibilités.

Pour l'instant, nous n'avons parlé que de la partie graphique de la page… Mais où est le code qui régit la cinématique de la page ? En l'occurrence, où et comment indique-t-on que les listes de familles et de produits doivent être remplies à partir des tables correspondantes de la base de données, et qu'un changement de sélection de la famille doit provoquer le rechargement de la liste des produits ? Nous allons répondre à cette question sans plus attendre, dans le paragraphe qui suit.

## Un code plus structuré grâce à la séparation du contenu HTML et de la cinématique

Avec les techniques de développement Web classique, le code qui régit le comportement de la page est *mélangé* au contenu HTML ; dans le cas d'une page ASP.NET, ce code est *séparé physiquement* du contenu HTML.

Le développeur dispose de deux techniques pour la localisation du code :

1. placer le code au sein du même fichier `.aspx` que le contenu HTML (*code in-line*) ;
2. placer le code dans un fichier séparé (*code behind*).

### Alternative 1 - Placer le code et le HTML de la page dans un même fichier

Intéressons-nous dans un premier temps à la première technique, illustrée à la figure 3-3, qui est celle que nous emploierons le plus souvent dans cet ouvrage. Elle correspond à un découpage du fichier `.aspx` en deux blocs distincts :

- le bloc « HTML », composé d'un mélange de balises HTML classiques et de balises spécifiques ASP.NET correspondant généralement à des contrôles serveur ;
- le bloc « code », encadré par deux balises `<script>`, qui contient l'implémentation du paramétrage des contrôles et de la cinématique de la page ;

**WEB MATRIX FACE À VS.NET**
**Gestion du code**

Une des différences principales entre Web Matrix et Visual Studio.NET se situe dans la technique de gestion du code : Web Matrix a choisi de placer le code et le contenu HTML dans un fichier unique, tout en offrant des vues logiques separées, grâce à un système d'onglet, alors que VS.NET a adopté la technique du code behind, probablement à cause de son analogie avec Visual Basic.

**SYNTAXE ASP.NET**
**Spécifier le langage utilisé**

La directive <%@ Page Language...%> permet, entre autres, de spécifier le langage utilisé dans la page (VB.NET, C# ou JScript.NET). En l'absence de cette directive, le langage par défaut est VB.NET, sauf si les paramètres par défaut ont été changés dans le fichier web.config de l'application.

**SYNTAXE ASP.NET**
**Position du bloc <script>**

Le bloc de code est généralement placé en haut du fichier .aspx, avant le bloc de contenu HTML. Néanmoins, il est techniquement possible de le placer où on le souhaite : par exemple en dessous, voire au milieu du bloc HTML. On peut même répartir le code entre plusieurs blocs, à condition que chacun soit encadré par des balises <script>.

Notons que ces deux blocs sont, en général, précédés d'une directive <% Page ...%>, facultative mais utilisée dans la majorité des cas, qui permet de spécifier un certain nombre d'attributs de la page.

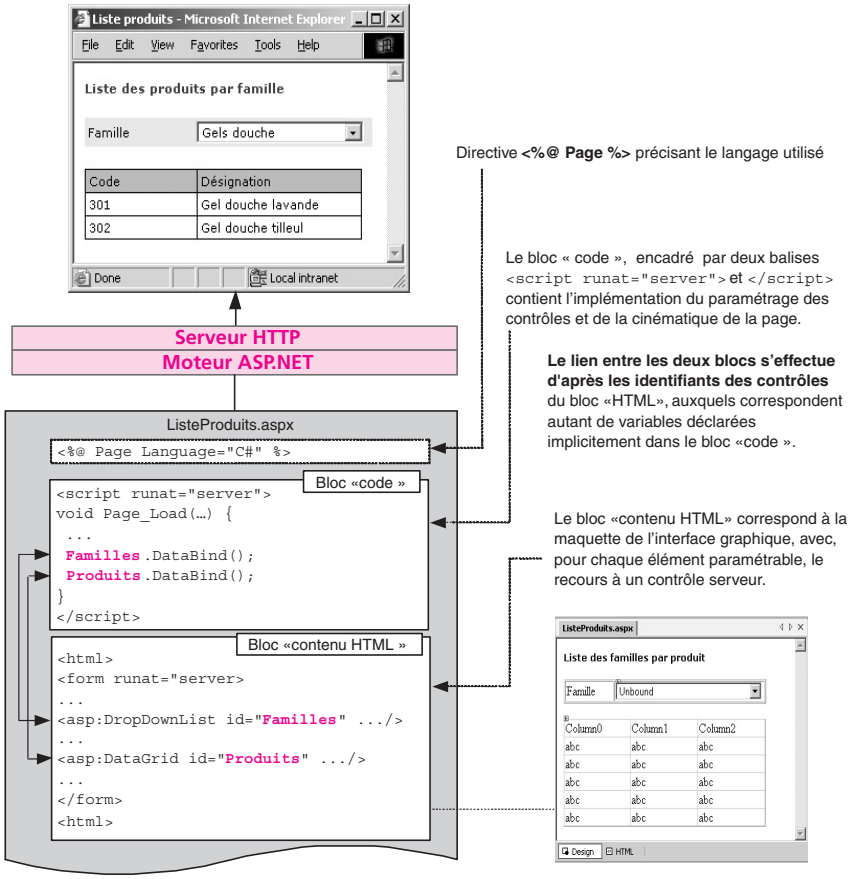



**Figure 3–3** Architecture d'une page ASP.NET

**ASP.NET Compatibilité avec ASP**

S'il est recommandé, pour une meilleure organisation, de placer l'intégralité du code de la page dans le bloc <script>, il est toutefois possible de placer du code au sein du contenu HTML, à l'intérieur d'un bloc <% …%>, du fait de la compatibilité ascendante avec ASP.

Sans entrer dans le détail du code, soulignons dès à présent deux points d'architecture fondamentaux :

- **Le lien entre entre les contrôles serveur et le code s'effectue par l'intermédiaire des identifiants** : à chaque contrôle serveur contenu dans le bloc « HTML » correspond une variable déclarée *implicitement* dans le bloc « code », le nom et le type de cette variable correspondant respectivement au nom et au type du contrôle.
- **Le code de la page est organisé en gestionnaires d'événements** : chaque portion de code s'exécute en réaction à un événement donné (ici, en l'occurrence, l'événement Page_Load, correspondant au chargement de la page).

Ce dernier principe nous permettra, par exemple, d'implémenter le rechargement de la liste des produits lorsque la sélection de la famille change, comme nous le verrons dans le prochain chapitre.

## Alternative 2 - Placer le code de la page dans un fichier séparé

Comme on l'a indiqué plus haut, une technique alternative (dite *code behind*) consiste à placer le code associé à la page dans un fichier distinct du fichier `.aspx` principal (figure 3-4).

Le principal avantage de cette séparation du contenu graphique et du code en deux fichiers distincts est de pouvoir faire travailler en parallèle un concepteur HTML et un développeur à la production d'une même page – à condition qu'ils se soient préalablement mis d'accord sur les types et les noms des contrôles.

### À quoi ça ressemble ?

Sans entrer dans le détail de la syntaxe, nous présentons ici l'extrait du code correspondant à l'affichage du contenu d'une table dans un tableau HTML – en l'occurrence, notre liste de produits – en comparant la version ASP et la version ASP.NET, afin que le lecteur puisse dès maintenant voir « à quoi ça ressemble ».

Dans la version ASP, les éléments HTML et les instructions en langage de script sont entremêlées :

**Version ASP**

```
<%
Set conn = Server.CreateObject("ADODB.Connection")
conn.Open "<Chaîne de connexion>"
Set rs = conn.Execute("SELECT * FROM Produit")
%>

<table border="1">
<%
Do While Not rs.EOF
   %>
   <tr>
      <td><%= rs.Fields("Code").Value %></td>
      <td><%= rs.Fields("Designation").Value %>
      </td>
   </tr>
   <%
   rs.MoveNext
Loop
%>
</table>
<%
   rs.Close
   conn.Close
%>
```

Dans la version ASP.NET, on note la séparation claire entre le code (entre les balises `<script>`) et la partie graphique (entre les balises `<html>`), ainsi que l'organisation du code en gestionnaires d'événements (ici, un seul gestionnaire : Page_Load) :

**Version ASP.NET (C#)**

```
<script runat="server">

void Page_Load(Object sender, EventArgs e)
{
   SqlConnection conn ;
   SqlCommand cmd;
   SqlDataReader reader;

   string SQL = "SELECT * FROM Produit";

   conn = new SqlConnection(<Chaîne de connexion>);
   conn.Open();

   cmd = new SqlCommand(SQL,conn);
   reader = cmd.ExecuteReader();

   Produits.DataSource = reader
   Produits.DataBind();
   reader.Close();
   conn.Close();
}

</script>

<html>
<asp:DataGrid id="Produits" runat="server"/>
</html>
```

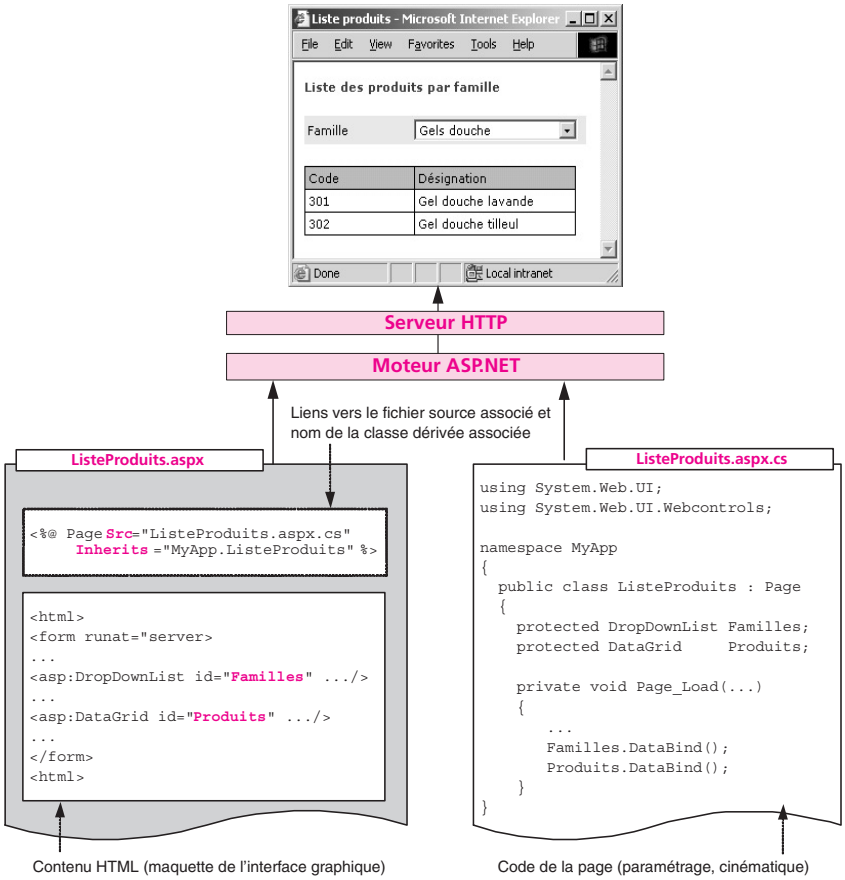



**Figure 3–4** Architecture d'une page ASP.NET, avec code dans un fichier à part

Un second avantage, plus théorique : cette option fait apparaître clairement la nature technique des pages ASP.NET car le fichier code associé doit contenir la définition d'une classe dérivée de `System.Web.UI.Page` ; il faut déclarer, *explicitement* cette fois, des variables membres correspondant aux contrôles serveur placés dans le fichier `.aspx`.

Par convention, on nomme généralement le fichier code associé à une page d'après le nom du fichier `.aspx`, auquel on ajoute l'extension propre au langage (`.vb` pour VB.NET, `.cs` pour C#) : par exemple, `MyPage.aspx.vb` (ou `.cs`) pour le code associé à `MyPage.aspx`.

Le lien entre le fichier `.aspx` et le code associé s'effectue par l'intermédiaire de l'attribut `Inherits` de la directive `<%@ Page %>`, qui doit obligatoirement spécifier le nom de la classe correspondant à la page. L'attribut `Src`, facultatif, permet d'indiquer au compilateur *Just-In-Time* le nom du fichier source, qui est alors compilé à la volée lors de l'exécution de la page ; en l'absence de directive `Src`, le moteur ASP.NET recherche l'implémentation de la classe associée à la page parmi les assemblages disponibles.

### Assemblage

Un assemblage (*assembly* en anglais) désigne une brique binaire élémentaire d'une application .NET. C'est en quelque sorte l'équivalent d'un composant COM dans le monde « Windows DNA » mais avec des fonctionnalités supplémentaires (regroupement logique de plusieurs fichiers DLL ou EXE dans un assemblage, gestion des versions, description de l'assemblage par des métadonnées…).

### ASP.NET Différences entre Inherits, Src, et Codebehind

Il existe parfois une confusion entre ces trois attributs, tous utilisés dans la directive `<%Page...%>` et relatifs au placement du code dans une page à part, mais dont les significations sont différentes :

- `Inherits` est le seul attribut qui soit obligatoire dans le cas de placement du code dans un fichier séparé. Il doit spécifier le nom de la classe associée à la page (dérivée de `System.Web.UI.Page`).
- `Src` est un attribut facultatif spécifiant le chemin du fichier source associé à la classe. Si cet attribut est spécifié, le moteur ASP.NET effectuera une compilation à la volée (*JIT compilation*) de la page indiquée. Dans le cas contraire, le moteur recherchera la classe associée à la page parmi les assemblages disponibles.
- `Codebehind` est le plus trompeur des trois : bien qu'il porte le nom dont on qualifie la technique de séparation du code et du contenu, ce n'est pas un attribut ASP.NET : c'est en réalité un attribut spécifique de Visual Studio.NET utilisé par l'éditeur HTML pour savoir quel est le fichier source associé au fichier graphique.

Cette technique a néanmoins pour inconvénient de rendre le déploiement plus complexe : nombre de fichiers plus important, nécessité de compiler à l'avance les sources si l'on n'utilise pas l'attribut `Src`. En outre, notons qu'il est difficile d'échanger des pages ASP.NET entre deux outils de développement n'ayant pas adopté la même technique de gestion du code – en l'occurrence, entre Visual Studio.NET et Web Matrix.

## Faciliter la réutilisation avec les contrôles utilisateur

La question de la réutilisation de « morceaux de pages » se pose fréquemment en développement Web : par exemple, on souhaite souvent placer une même barre de navigation sur toutes les pages d'un site ou utiliser un même composant (boîte de dialogue Login ou Recherche) dans plusieurs sites.

Pour faire cela avec une technologie classique comme ASP, il fallait utiliser une instruction de type `<!--#include file… -->` permettant d'inclure un fichier à un endroit donné. Ce mécanisme permettait, dans une certaine mesure, d'organiser le code mais présentait un certain nombre d'inconvénients : en particulier, il y avait risque de *collisions de noms* entre les variables du fichier principal et du fichier inclus.

Pour pallier ce type d'inconvénients, ASP.NET a introduit la notion de *contrôle utilisateur*, autrement dit de *morceau de page Web réutilisable* : du point de vue du développeur, un contrôle utilisateur s'implémente, à quelques détails près, comme une page ASP.NET normale ; du point de vue de la page utilisatrice, ce contrôle est vu comme un objet externe.

La figure 3-5 illustre l'utilisation d'un contrôle utilisateur – une barre de navigation :

- le contrôle doit être implémenté dans un fichier d'extension `.ascx` et doit comporter une directive `<% Control…%>` à la place de la directive `<% Page…%>` ;
- la page utilisatrice doit enregistrer le contrôle avec une directive `<%Register %>`, qui précise le nom du contrôle (ici : `Header`) et son espace de nommage (ici : `SDS` pour « Savons du Soleil ») ;
- enfin, il peut être fait référence au contrôle, à l'endroit où l'on souhaite l'insérer, comme pour un contrôle serveur normal : `<SDS:Header…runat="server">`.

Voici les principaux avantages offerts par les contrôles utilisateur :

- les noms utilisés par le contrôle sont *encapsulés* et ne risquent pas d'entrer en collision avec ceux de la page utilisatrice ;
- le contrôle est un objet pouvant exposer des propriétés : par exemple, l'index de la rubrique active d'une barre de navigation peut être paramétré depuis l'extérieur ;
- les contrôles sont organisés dans des espaces de nommage : plusieurs contrôles peuvent porter le même nom s'ils sont dans des espaces de nommage différents.

**Contrôle utilisateur**

Morceau de page Web encapsulé dans un composant graphique réutilisable. Très faciles à implémenter, les contrôles utilisateur permettent la réutilisation d'éléments graphiques constitutifs d'une application (barre de navigation, menu, etc.) et remplacent en quelque sorte les `<!-- #Include -->` des pages ASP).

**Pagelets**

Les contrôles utilisateur (*user controls*) sont parfois dénommés *pagelets*, que l'on peut approximativement traduire par « morceaux de page ».

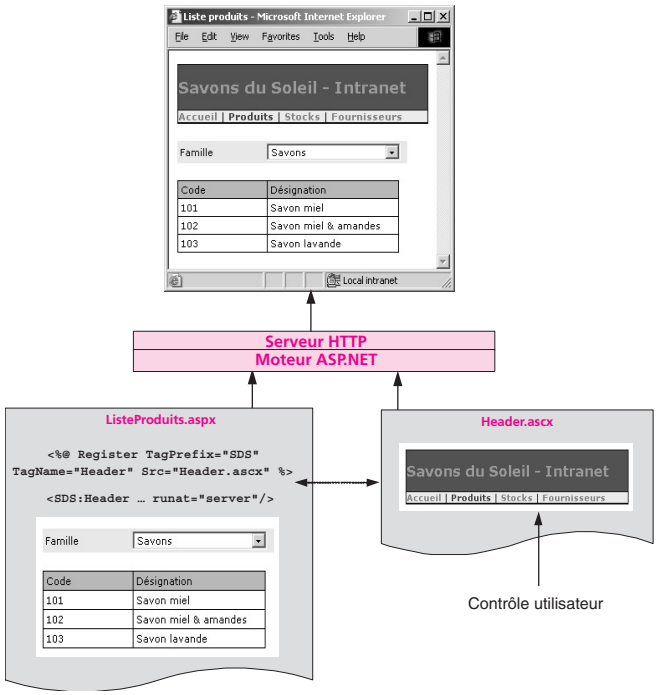



**Figure 3–5** Utilisation d'un contrôle utilisateur

Il est à présent temps de mettre en pratique ces notions de contrôle serveur et contrôle utilisateur, en réalisant la barre de navigation qui sera utilisée dans notre étude de cas.

# Mise en pratique : réalisation d'une barre de navigation

Dans cette seconde partie de chapitre, nous allons réaliser un contrôle utilisateur de type « barre de navigation », qui nous servira pour la suite de l'étude de cas : après une rapide prise en main de l'environnement de développement Web Matrix, nous réaliserons en deux temps la barre de navigation – partie graphique et implémentation - puis nous verrons comme l'intégrer au sein d'une page existante, en mettant en place le squelette de notre intranet.

## Création de la barre de navigation

Notre projet est donc de réaliser une barre de navigation semblable à celle illustrée à la figure 3-6, qui sera présente dans toutes les pages de l'intranet et permettra à l'utilisateur de naviguer d'une rubrique à l'autre ; cette barre de navigation devra afficher le titre de la rubrique en cours et mettre en surbrillance la rubrique active.

**Figure 3–6** Le projet de barre de navigation

Voyons maintenant comment utiliser Web Matrix pour créer ce contrôle utilisateur.

## Création d'un contrôle utilisateur avec Web Matrix

Démarrez Web Matrix (les instructions d'installation se trouvent dans le chapitre précédent) :

- une boîte de dialogue apparaît, proposant différents types de fichiers (figure 3-7) ;
- choisissez ASP.NET User Control ;
- indiquez votre répertoire de travail dans le champ Location ;
- spécifiez un nom de fichier, par exemple : `NavBar.ascx` ;
- enfin, indiquez le langage que vous souhaitez utiliser (C# ou VB.NET).

Une fois la page créée, l'interface principale de Web Matrix apparaît (figure 3-8).

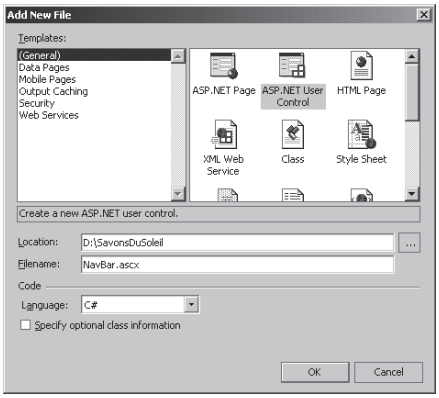



**Figure 3–7** Choix du type de fichier dans Web Matrix

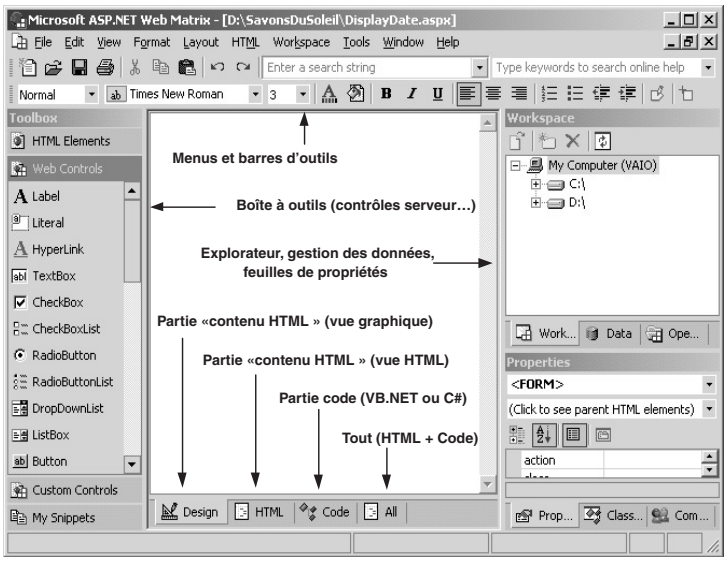



**Figure 3–8** L'interface principale de Web Matrix

**WEB MATRIX**
**Pas de Undo en mode design**

À l'heure actuelle, la fonctionnalité Undo est inactive dans l'onglet Design de Web Matrix. Peut-être cette fonctionnalité sera-t-elle implémentée dans la version finale ?

La partie centrale de l'interface est occupée par un éditeur de texte, qui fait apparaître clairement la séparation entre contenu graphique et code grâce à quatre onglets :

- l'onglet Design permet une édition du contenu HTML en mode graphique ;
- l'onglet HTML contient le code HTML correspondant ;
- l'onglet Code contient le code ASP.NET de la page (VB.NET ou C#) ;
- l'onglet All affiche le fichier complet, tel qu'il est stocké sur le disque.

De part et d'autre de cette zone centrale sont situées un certain nombre de fenêtres utilitaires :

- boîtes à outils « contrôles » sur la gauche ;
- explorateurs (fichiers, données) et feuilles de propriétés sur la droite.

Pour l'instant, tous les onglets sont vides, à l'exclusion de l'onglet All qui commence par une directive destinée à indiquer au moteur ASP.NET que le contenu du fichier correspond à un contrôle utilisateur :

```
<%@ Control Language="<VotreLangage>" %>
```

Passons maintenant à la réalisation de notre barre de navigation, qui s'effectuera, comme il se doit, en deux temps : réalisation de la partie graphique puis implémentation du code.

## Réalisation de la partie graphique de la barre de navigation

**RAPPEL Télécharger le code source**

Tout le code source de l'application, ainsi que les divers fichiers annexes, dont les images, sont disponibles en téléchargement à l'adresse :
▸ http://www.savonsdusoleil.com

Pour commencer, nous allons réaliser la partie graphique de notre barre de navigation, en nous plaçant dans l'onglet Design de Web Matrix.

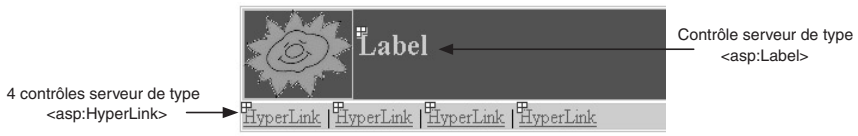



**Figure 3–9** Partie graphique de la barre de navigation

D'un point de vue HTML, notre contrôle est une table comportant deux lignes :

- la première contiendra le logo de la société et le titre ;
- la seconde les liens HTML vers les rubriques.

Le titre et les liens devant être paramétrables en fonction de la rubrique, nous allons les implémenter avec des *contrôles serveur*, de type `<asp:Label>` pour le titre et de type `<asp:Hyperlink>` pour les liens.

La marche à suivre pour insérer une table est très simple :

1. choisissez Insert Table dans le menu HTML : une boîte de dialogue apparaît (figure 3-10) ;
2. spécifiez le nombre de lignes et de colonnes (en l'occurrence : 2 lignes, 1 colonne).

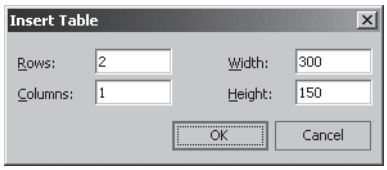



**Figure 3–10** Insérer une table

Malheureusement, les possibilités de paramétrage de la taille du tableau sont relativement réduites : la boîte de dialogue n'accepte que des valeurs en pixels. Dans notre cas, nous souhaitons que la table occupe 100% de la largeur de la page et que la hauteur ne soit pas spécifiée.

Pour cela, deux solutions sont possibles :

- utiliser la boîte de dialogue Propriétés pour modifier le style de la table (figure 3-11) ;
- modifier directement le contenu HTML (`<table style="WIDTH:100%">`).

Nous allons maintenant placer une image dans la première ligne de la table. Pour cela :

1. sélectionnez l'élément Image dans la boîte à outils Eléments HTML (figure 3-12) et le faire glisser vers la première ligne de la table ;
2. utilisez la boîte de dialogue Propriétés pour spécifier la source de l'image (`soleil.gif`) ;
3. spécifiez également la valeur `center` pour l'attribut `align`.

Enfin, spécifiez deux couleurs différentes pour les deux lignes de la table, par exemple :

- couleur verte pour la ligne supérieure (logo et titre) : `bgcolor="#00c000"` ;
- couleur jaune pour la ligne inférieure (rubriques) : `bgcolor="#ffff80"`.

Nous en avons fini avec la partie purement HTML de l'interface graphique ; nous allons maintenant passer à la mise en place des contrôles serveur.

Notre barre de navigation fera appel à deux types de contrôles serveur :

- un contrôle de type `Label` pour le titre ;
- quatre contrôles de type `HyperLink` pour les liens vers les rubriques.

Pour créer ces contrôles, on utilisera la boîte à outils Contrôles Web (figure 3-13) qui, lorsqu'on sélectionne puis place un contrôle, génère automatiquement la balise `<asp :TypeControle...>` correspondante dans le contenu HTML.

Commençons par le titre :

1. sélectionnez un contrôle serveur de type `Label` dans la barre d'outils ;
2. faites-le glisser vers la première ligne de la table, à droite du soleil (voir figure 3-9) ;
3. ajustez les caractéristiques du contrôle à l'aide des barres d'outils : gras, taille de police « 5 » et couleur jaune.

Plaçons maintenant les liens hypertexte :

1. sélectionnez un contrôle serveur de type `Hyperlink` dans la barre d'outils ;
2. faites-le glisser vers la deuxième ligne de la table (voir figure 3-9) ;
3. répétez trois fois l'opération, en séparant les contrôles par une barre verticale (« | »).

Servez-vous de l'onglet HTML de Web Matrix pour voir le code qui a été généré : une balise `<asp:TypeControle id="NomControle" runat="server">` a été créée pour chaque contrôle serveur.

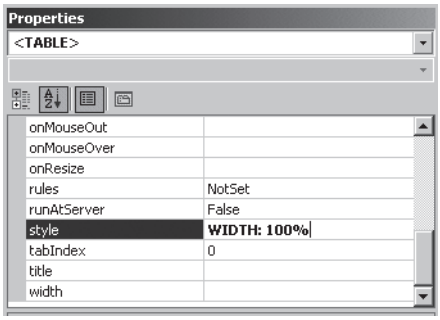



**Figure 3–11** Modifier les propriétés de la table

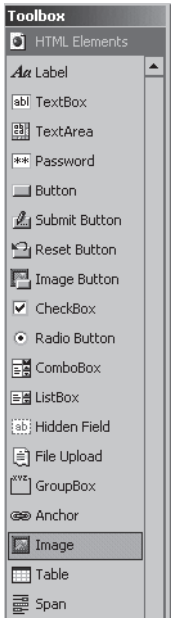



**Figure 3–12** La boîte à outils Eléments HTML

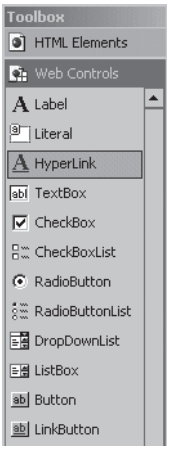



**Figure 3–13** La boîte à outils Contrôles Web

**ALTERNATIVE Utilisation d'un autre éditeur HTML**

Si vous le préférez, il est tout à fait possible de réaliser le contenu HTML à l'aide d'un autre éditeur plus complet (Dreamweaver, FrontPage…), puis de revenir à Web Matrix pour placer les contrôles serveur et implémenter le code.

**TERMINOLOGIE Contrôles Web ou contrôles serveur ?**

Web Matrix regroupe sous le vocable de « contrôle Web » l'ensemble des contrôles serveur, à l'exclusion des contrôles mobiles, tandis que, dans la documentation .NET, le terme « contrôles Web » se réfère parfois uniquement aux équivalents des contrôles HTML standards (boutons, liens, zones de texte…).

**Changer le nom des contrôles avec Web Matrix**

Il existe deux moyens pour changer le nom d'un contrôle avec Web Matrix : modifier la rubrique (ID) dans la feuille de propriétés du contrôle, ou modifier directement la balise correspondante dans l'onglet HTML.

Les noms de contrôles générés par défaut sont de type `HyperLink1`, `HyperLink2`, etc. Pour plus de clarté, renommez-les conformément au tableau ci-dessous :

| Nom du contrôle | Type | Rôle |
|---|---|---|
| Titre | `<asp:Label>` | Titre de la rubrique |
| Rubrique1 | `<asp:HyperLink>` | Lien vers la rubrique « Accueil » |
| Rubrique2 | `<asp:HyperLink>` | Lien vers la rubrique « Stocks » |
| Rubrique3 | `<asp:HyperLink>` | Lien vers la rubrique « Fournisseurs » |
| Rubrique4 | `<asp:HyperLink>` | Lien vers la rubrique « Ventes » |

Voici, après renommage des contrôles, le contenu HTML de notre barre de navigation.

**NavBar.ascx (partie graphique)**

```
<table style="WIDTH: 100%">
  <tbody>
    <tr>
      <td bgcolor="#00c000">
        <img src="../img/soleil.gif" align="center" />
        <asp:Label id="Titre" runat="server" Font-Bold="True"
         Font-Size="Large" ForeColor="#ffff80"/>
      </td>
    </tr>
    <tr>
      <td bgcolor="#ffff80">
        <asp:HyperLink id="Rubrique1" runat="server"/> |
        <asp:HyperLink id="Rubrique2" runat="server"/> |
        <asp:HyperLink id="Rubrique3" runat="server"/> |
        <asp:HyperLink id="Rubrique4" runat="server"/> |
      </td>
    </tr>
  </tbody>
</table>
```

Et voilà, la partie graphique de notre contrôle utilisateur est terminée !

Nous allons maintenant passer dans l'onglet **Code** pour implémenter le paramétrage et la cinématique de notre barre de navigation.

## Programmation de la barre de navigation

Lors du chargement de la barre de navigation, il faut effectuer les opérations suivantes :

- paramétrage des liens HTML vers les rubriques ;
- mise à jour du titre en fonction de la rubrique sélectionnée ;
- mise en surbrillance de la rubrique sélectionnée.

L'utilisateur de la barre de navigation doit avoir un moyen de spécifier la rubrique sélectionnée : ceci se fera par l'intermédiaire d'une *propriété* exposée par le contrôle utilisateur.

Pour réaliser la mise en surbrillance de la rubrique active, nous allons faire appel à une feuille de style, définissant une classe `active-rubrique` :

**SDS.css (HTML)**

```
body      { font-family: Verdana }
p         { font-size: 8pt  }
table     { font-size: 8pt }
h4        { font-size: 8pt; font-weight:bold; color: #000080 ; }
a         { font-size: 8pt; text-decoration:none; color: #000080 }
a.active-rubrique    { font-weight:bold }
```

Cette feuille de style (SDS comme « Savons du Soleil ») permet de définir quelques caractéristiques de la page : utilisation de la police Verdana (taille 8 points), liens en bleu non soulignés, lien en bleu gras pour la rubrique active. Avant de continuer, créez cette feuille de style avec un éditeur de texte ou avec Web Matrix et sauvegardez-la dans votre répertoire de travail.

Passons maintenant au code du contrôle, dont nous présentons ici deux versions, VB.NET et C# :

**NavBar.ascx (code) – Version C#**

```
public int SelectedIndex;
void Page_Load(Object sender, EventArgs e)
{
    Rubrique1.Text = "Accueil";
    Rubrique1.NavigateUrl = "default.aspx";
    Rubrique2.Text = "Stocks";
    Rubrique2.NavigateUrl = "stocks.aspx";
    Rubrique3.Text = "Fournisseurs";
    Rubrique3.NavigateUrl = "fournisseurs.aspx";
    Rubrique4.Text = "Ventes";
    Rubrique4.NavigateUrl = "ventes.aspx";
    switch (SelectedIndex)
    {
        case 0 :
            Titre.Text = "Accueil Intranet";
            Rubrique1.CssClass = "active-rubrique";
        break;
        case 1 :
            Titre.Text = "Suivi des stocks";
            Rubrique2.CssClass = "active-rubrique";
        break;
        case 2 :
            Titre.Text = "Gestion des fournisseurs";
            Rubrique3.CssClass = "active-rubrique";
        break;
        case 3 :
            Titre.Text = "Analyse des ventes";
            Rubrique4.CssClass = "active-rubrique";
        break;
    }
}
```

Propriété publique spécifiant la rubrique active

Mise à jour des liens

Mise à jour du titre

**La propriété CssClass**

Cette propriété permet de paramétrer la classe de style (CSS : *Cascading Style Sheet*) d'un contrôle serveur et correspond à l'attribut `class` d'une balise HTML. En l'occurrence, dans notre exemple, le contenu HTML généré pour le lien actif sera le suivant : `<a class="active-rubrique"...>`

Pour plus d'information sur les feuilles de style :

- http://www.w3.org/style/css

**NavBar.ascx (code) – Version VB.NET**

Propriété publique spécifiant la rubrique active

Mise à jour des liens

Mise à jour du titre

```
Public SelectedIndex As Int32
Sub Page_Load(sender As Object,e As EventArgs)
    Rubrique1.Text = "Accueil"
    Rubrique1.NavigateUrl = "default.aspx"
    Rubrique2.Text = "Stocks"
    Rubrique2.NavigateUrl = "stocks.aspx"
    Rubrique3.Text = "Fournisseurs"
    Rubrique3.NavigateUrl = "fournisseurs.aspx"
    Rubrique4.Text = "Ventes"
    Rubrique4.NavigateUrl = "ventes.aspx"
    Select Case SelectedIndex
        Case 0
            Titre.Text = "Accueil Intranet"
            Rubrique1.CssClass = "active-rubrique"
        case 1
            Titre.Text = "Suivi des stocks"
            Rubrique2.CssClass = "active-rubrique"
        case 2
            Titre.Text = "Gestion des fournisseurs"
            Rubrique3.CssClass = "active-rubrique"
        case 3
            Titre.Text = "Analyse des ventes"
            Rubrique4.CssClass = "active-rubrique"
    End Select
End Sub
```

L'examen de ce code est relativement rapide.

On commence par déclarer une propriété publique `SelectedIndex`, qui sera accessible depuis l'extérieur du contrôle et permettra à l'utilisateur de spécifier l'index de la rubrique active (entre 0 et 3) depuis la page qui incluera le contrôle.

Puis, dans le gestionnaire `Page_Load` exécuté lors du chargement de la page, on paramètre les liens hypertextes – mise à jour du texte et de l'adresse de lien via les propriétés `Text` et `NavigateUrl` – on spécifie le titre – mise à jour de la propriété `Text` du contrôle `Titre` – puis on attribue à la rubrique active le style `active-rubrique`, défini dans la feuille de style grâce la propriété `CssClass`.

Notre barre de navigation est maintenant terminée. Néanmoins, il n'est pas possible de la tester individuellement : il nous faut obligatoirement l'intégrer dans une page ASP.NET. C'est ce que nous allons faire dans le paragraphe qui suit en créant le squelette de notre intranet.

**À propos de Page_Load**

Dans le chapitre suivant, nous aurons l'occasion de parler plus amplement des événements liés à une page : `Page_Init`, `Page_Load`, etc. Dans un premier temps, retenons uniquement que `Page_Load` est appelé lors du chargement de la page.

**Aparté pour les programmeurs objet**

L'objet exécutable qui produit le code HTML correspondant à cette page est une instance d'une classe dérivée de `System.Web.UI.Page`. `Page_Load` est une fonction virtuelle redéfinie dans cette classe dérivée. Enfin, les contrôles serveur sont implicitement déclarés comme des variables membres (propriétés) privées .

## Création du squelette de l'intranet

Notre intranet sera composé de quatre rubriques principales :

- page d'accueil ;
- suivi des stocks ;

- gestion des fournisseurs ;
- analyse des ventes.

La barre de navigation devra apparaître en haut de chacune des pages, la rubrique active étant sélectionnée (voir figure 3-14).

Commençons par réaliser la page d'accueil : pour cela, créer dans Web Matrix un nouveau fichier nommé `default.aspx`, en choisissant cette fois le type `ASP.NET Page` (c'est le choix par défaut).

L'intégration de la barre de navigation en haut de la page se fait en deux étapes :

1. enregistrement du contrôle utilisateur avec l'instruction `<% Register…%>` ;
2. positionnement du contrôle dans la page.

L'enregistrement du contrôle, indispensable à son utilisation dans la page, s'effectue à l'aide de l'instruction suivante, placée en haut du fichier (juste après la directive `<%@ Page …%>`) :

```
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
```

Les significations des différents attributs, tous obligatoires, sont les suivantes :

- `TagPrefix` indique le préfixe qui sera utilisé pour référencer le contrôle, autrement dit l'espace de nommage auquel il appartient ;
- `TagName` spécifie le nom du contrôle ;
- `Src` indique le nom du fichier source correspondant au contrôle.

L'enregistrement effectué, on peut maintenant inclure la balise correspondant au contrôle, à l'endroit où l'on souhaite que la barre de navigation soit insérée :

```
<SDS:NavBar id="MyNavBar" SelectedIndex="0" runat="server">
</SDS:NavBar>
```

On note évidemment la similitude avec une balise de contrôle serveur, la différence étant qu'on a remplacé `<asp:TypeControle…>` par `<TagPrefix:TagName…>`.

Autre point important, *la propriété publique* `SelectedIndex`, déclarée dans le code du contrôle, *est accessible sous forme d'attribut de la balise du contrôle* ; dans notre cas, l'index correspondant à la page d'accueil est 0.

Terminons la réalisation de notre page d'accueil en insérant un lien vers la feuille de style externe `SDS.css` définie plus haut, indispensable pour la mise à jour de la rubrique active :

```
<link href="SDS.css" type="text/css" rel="stylesheet" />
```

Enfin, pour remplir la page, on se propose d'ajouter des liens vers les autres rubriques dans le corps de la page.

**Figure 3–14** Page d'accueil de l'intranet

### À propos de TagPrefix

`TagPrefix` contient généralement le nom de l'application ou le nom de la société (ici : SDS pour « Savons du Soleil »). Le principal intérêt de cet attribut est de permettre l'existence de plusieurs contrôles portant le même nom, à condition qu'ils soient dans des espaces de nommage distincts.

### Le nom du contrôle est fixé depuis la page cliente

Notons que le nom de la classe du contrôle est fixé par la page utilisatrice, via les attributs `TagPrefix` et `TagName`, contrairement à ce qui se passe dans une interface classique, de type Visual Basic, où seul le nom de la variable correspondant au contrôle est paramétrable par l'utilisateur.

Voici le code complet de la partie graphique de la page d'accueil :

**default.aspx (partie graphique)**

```
<%@ Page Language="<VotreLangage>" Debug="true" %>
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
<html>
<head>
    <title>Savons du Soleil - Intranet</title>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <SDS:NavBar id="MyNavBar" SelectedIndex="0" runat="server">
    </SDS:NavBar>
    <p></p><p></p>
    <form runat="server">
        <h4>Bienvenue sur l'intranet des Savons du soleil
        </h4>
        <p><a href="./Stocks.aspx">Suivi des stocks</a> </p>
        <p><a href="./Fournisseurs.aspx">Gestion des fournisseurs</a>
        </p>
        <p><a href="./Ventes.aspx">Analyse des ventes</a></p>
    </form>
</body>
</html>
```

Il reste à répéter trois fois l'opération pour les autres rubriques, en modifiant uniquement le titre de la page, le nom du fichier et la valeur de `SelectedIndex`.

| Rubrique | Nom du fichier | Valeur de SelectedIndex |
|---|---|---|
| Suivi des stocks | `stocks.aspx` | 1 |
| Gestion des fournisseurs | `fournisseurs.aspx` | 2 |
| Analyse des ventes | `ventes.aspx` | 3 |

À titre d'exemple, voici le code de la partie graphique du fichier `stocks.aspx` :

**stocks.aspx (partie graphique)**

```
<%@ Page Language="<VotreLangage>" Debug="true" %>
<%@ Register TagPrefix="SDS" TagName="NavBar" Src="NavBar.ascx" %>
<html>
<head>
    <title>Suivi des stocks</title>
    <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <SDS:NavBar id="MyNavBar" runat="server" SelectedIndex="1">
    </SDS:NavBar>
    <p></p> <p></p>
    <form runat="server">
        <h4>Bienvenue dans le module de suivi des stocks </h4>
    </form>
</body>
</html>
```

Le développement du squelette de notre application est terminé : il ne reste plus qu'à le tester !

## Tester l'application

Pour tester notre application, deux options sont possibles :

- utiliser le serveur HTTP léger fourni avec Web Matrix ;
- utiliser le serveur IIS.

### À propos de la compilation des pages ASP.NET

Si vous avez fait une erreur dans la syntaxe du code (par exemple, vous avez omis le « ; » final d'une instruction en C#), une erreur de compilation se produira : vous verrez alors apparaître un écran similaire à celui représenté à la figure 3-15.

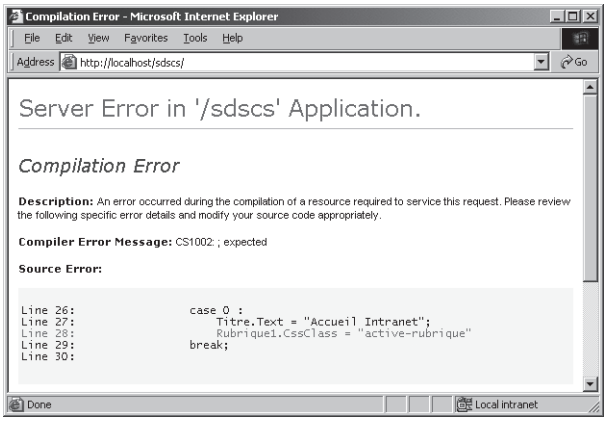



**Figure 3–15** Erreur de compilation

Ceci permet au passage de souligner un des avantages du caractère *compilé* des pages ASP.NET : l'*intégralité* du code est passée en revue avant la première exécution de la page et *toutes* les erreurs de syntaxe sont détectées lors de cette phase de compilation, alors qu'avec une page interprétée (de type ASP), seules les erreurs dans les éléments interprétés étaient détectées (à l'exclusion, par exemple, des erreurs dans les branches non exécutées du code).

Néanmoins, la compilation n'élimine pas totalement le risque d'erreur à l'exécution, des *exceptions* pouvant toujours se produire.

Le résultat de la compilation de la page est un fichier exécutable – un assemblage dans le langage .NET – stocké sous la forme d'une DLL dans le répertoire Temporary ASP.NET Files situé sous le répertoire Microsoft.NET dans le dossier système (figure 3-16).

Si vous jetez un œil à ce dossier – ou plus précisément dans celui des ses sous-dossiers qui porte le nom de votre application – vous y trouverez les fichiers `.dll` correspondant aux résultats de la compilation de

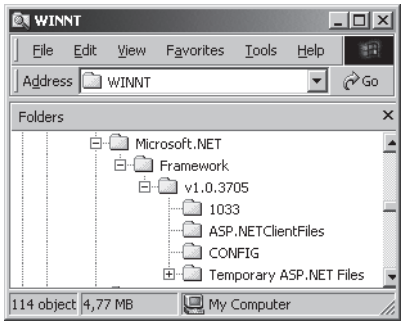



**Figure 3–16** Le dossier Temporary ASP.NET Files

vos pages : chacun de ces fichiers est, en quelque sorte, un exécutable capable de « cracher » le HTML correspondant à la page ou au contrôle utilisateur.

Ces fichiers contiennent du langage IL (*Intermediate Language*) qu'il est possible de désassembler en utilisant l'utilitaire `ildasm.exe` situé dans le répertoire FrameworkSDK (figure 3-17).

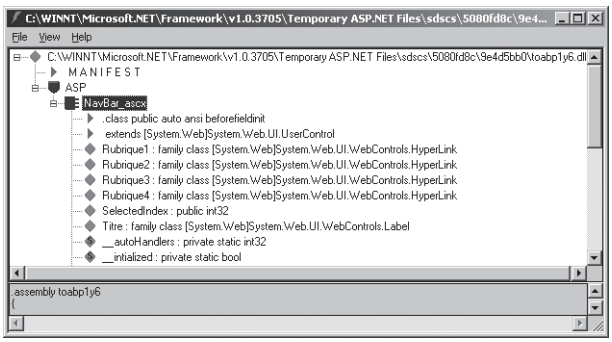



**Figure 3–17** Désassemblage de la barre de navigation

Rappelons pour finir que la compilation n'intervient que lors de la première requête vers la page ; lors de requêtes ultérieures, l'assemblage en cache est réutilisé, sauf si le code source a été modifié entre temps.

Positionnez-vous sur la page d'accueil et sélectionnez Start dans le menu View, ou utilisez la touche de raccourci F5, pour faire apparaître la boîte de dialogue permettant d'effectuer le choix du serveur HTTP (figure 3-18).

La première option permet de sélectionner le serveur Web Matrix (figure 3-19) qui a pour principal avantage d'éviter l'installation d'IIS, bien qu'il soit plus limité fonctionnellement (pas de racines virtuelles, de pages d'erreurs personnalisées, etc.).

La seconde option permet d'utiliser le serveur IIS : dans la case Application Name, saisissez le nom de la racine virtuelle de votre application. Si elle n'existe pas, Web Matrix la créera.

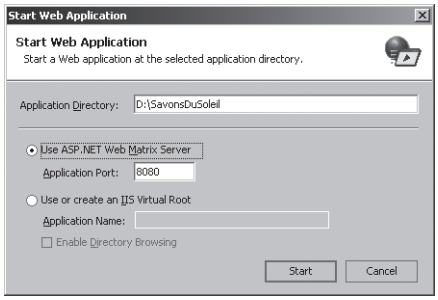



**Figure 3–18** Choix du serveur HTTP

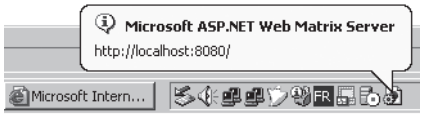



**Figure 3–19** Démarrage du serveur Web Matrix

## En résumé…

Dans ce long chapitre, nous avons présenté les principes fondamentaux de la technologie ASP.NET :

- le *modèle de programmation orienté interface* qui permet d'appréhender une page Web comme une interface graphique classique : un assemblage de contrôles graphiques réagissant à des événements ;
- la *séparation entre partie graphique et code,* qui rend le contenu HTML plus clair car il n'est plus entremêlé avec des scripts et peut faciliter, le cas échéant, la répartition du travail entre développeurs et concepteurs (designers) ;
- les *contrôles serveur*, qui augmentent la productivité du développement en encapsulant la génération de HTML et peuvent réagir à des événements ;
- les *contrôles utilisateur* qui constituent un moyen efficace et simple de réutilisation de composants graphiques.

Puis nous avons mis en œuvre ces mécanismes pour réaliser la maquette de notre intranet, en effectuant au passage un prise en main de l'environnement de développement Web Matrix.

Dans le chapitre suivant, nous allons implémenter le module de suivi des stocks de notre application, qui sera l'occasion d'illustrer deux mécanismes importants :

- l'accès à une base de données depuis une application ASP.NET ;
- la gestion des événéments côté serveur.

# Consulter une base de données : l'interface de suivi des stocks

# 4

ASP.NET

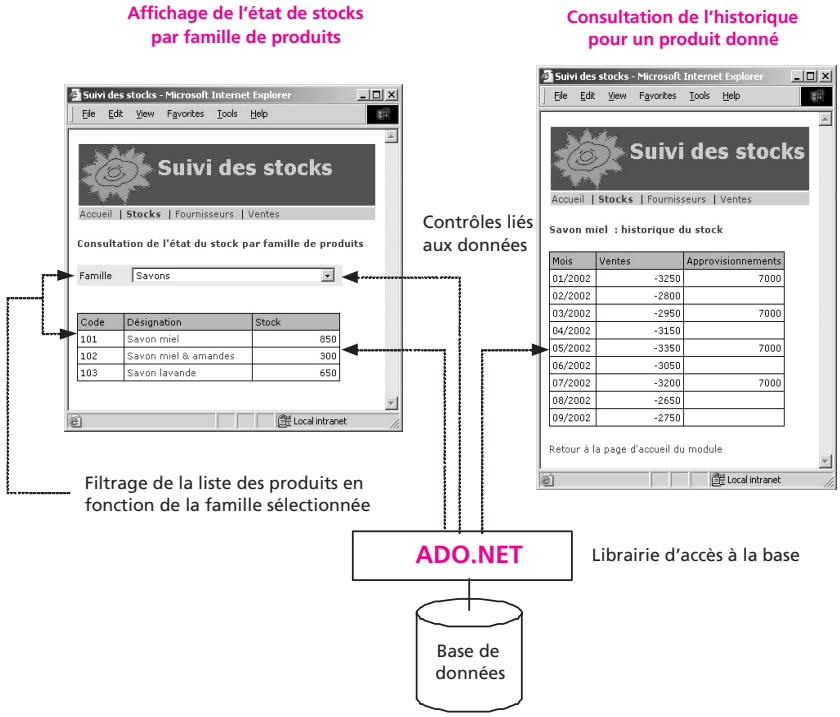
ADO.NET | DataGrid | DropDownList | ViewState | IsPostBack | global.asax | Session

## SOMMAIRE

- Présentation de la bibliothèque ADO.NET
- Utilisation de contrôles liés aux données
- Gestion des événéments

## MOTS-CLÉS

- ADO.NET
- DataGrid
- DropDownList
- SqlConnection
- SqlCommand
- SqlDataReader
- SqlDataAdapter
- ViewState
- IsPostBack
- global.asax
- Session




Dans ce chapitre, nous réalisons l'interface de suivi des stocks, qui repose principalement sur l'utilisation de contrôles serveur (`DataGrid` et `DropDownList`) liés à la source de données par l'intermédiaire de la bibliothèque ADO.NET, que nous présentons au passage. Nous abordons également le mécanisme des événements serveur, lequel nous permettra d'ajouter de l'interactivité à notre interface.

# Réalisation de la maquette de l'interface

Le module de suivi des stocks doit permettre, d'une part, l'affichage de la synthèse des quantités en stock, filtrées par famille de produits et, d'autre part, la consultation de l'historique des variations de stocks pour un produit donné.

Notre interface sera, par conséquent, constituée de deux pages distinctes :

- La page `stocks.aspx`, écran d'accueil du module, présentera la liste des stocks disponibles pour une famille de produits donnée. Il devra être possible de sélectionner la famille désirée dans une liste déroulante, la liste étant alors mise à jour en conséquence ; et il devra également être possible de cliquer sur un produit donné pour accéder à l'historique du stock de ce produit.
- La page `detailstock.aspx` présentera l'historique des stocks d'un produit donné (détail des ventes et approvisionnement par mois).

Comme pour toute page ASP.NET, le développement se déroulera en deux temps :

- réalisation de la maquette de la page ;
- implémentation de la cinématique de la page.

Nous allons donc commencer par réaliser la maquette de ces deux pages.

## Maquette de la page de consultation des stocks par famille de produits

À la fin du chapitre précédent, nous avons réalisé le squelette de la page d'accueil du module de suivi des stocks (fichier `stocks.aspx`), dans laquelle nous avons déjà intégré la barre de navigation.

Nous allons repartir de cette ébauche, à laquelle nous allons ajouter deux contrôles serveur :

- un contrôle de type `DropDownList` (liste déroulante) pour afficher la liste des familles ;
- un contrôle de type `DataGrid` (grille de données) pour afficher l'état des stocks.

Le résultat auquel nous parviendrons lorsque la page sera réalisée est représenté figure 4-1.

Avant de modifier notre page, nous allons en sauvegarder une copie, qui nous servira également de point de départ pour la page de consultation détaillée de l'historique :

1. Démarrez Web Matrix.
2. Ouvrez la version actuelle du fichier `stocks.aspx`.
3. Sauvegardez-la sous le nom `detailstocks.aspx`, dans le même répertoire.

Ceci étant fait, nous pouvons passer à la réalisation de la maquette de notre première page :

1. Ouvrez à nouveau la version actuelle du fichier `stocks.aspx`.

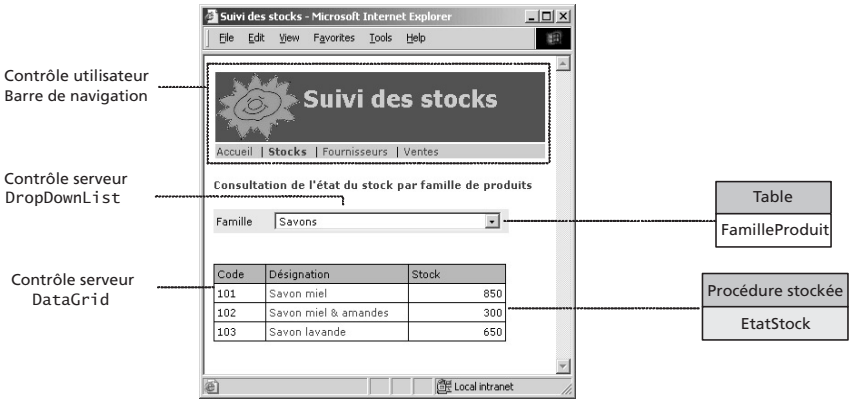



**Figure 4–1** La page de consultation des stocks par famille de produits

**2** Placez-vous dans l'onglet Design de Web Matrix.

**3** Insérez un titre (par exemple : « Consultation de l'état du stock par famille de produits »).

**4** Créez un tableau HTML (une ligne ; deux colonnes, arrière-plan jaune).

**5** Insérez un contrôle serveur `DropDownList` dans la partie droite de ce tableau et renommez-le `ListeFamilles`.

**6** Insérez un contrôle serveur `DataGrid` sous le tableau et renommez-le `EtatStock`.

Le résultat de la maquette doit ressembler à celle montrée figure 4-2.

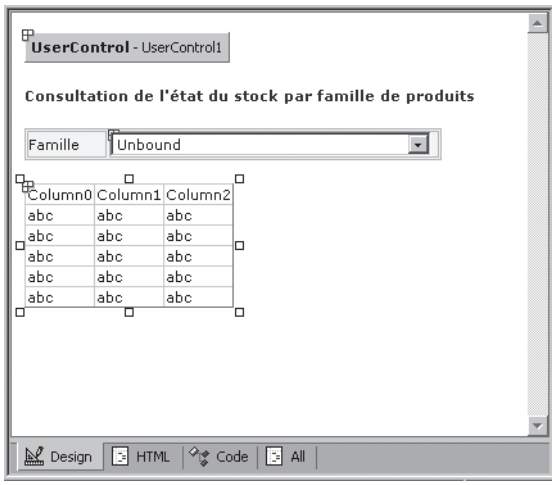



**Figure 4–2** Maquette de la page de suivi des stocks

**RAPPEL Insérer un contrôle serveur**

Pour insérer un contrôle serveur, il faut utiliser l'onglet Web Controls de Web Matrix et faire glisser les éléments voulus aux endroits souhaités.

**ATTENTION Ne pas confondre DataGrid et MxDataGrid**

Nous utiliserons dans notre étude de cas le contrôle serveur standard `DataGrid`, à ne pas confondre avec le contrôle serveur spécifique nommé `MxDataGrid`, fourni avec Web Matrix, qui offre également des fonctionnalités d'affichage de grilles de données, mais que nous n'étudierons pas dans cet ouvrage.

**RAPPEL Renommer un contrôle**

Il existe deux moyens pour changer le nom d'un contrôle avec Web Matrix : modifier la rubrique (ID) dans la feuille de propriétés du contrôle, ou modifier directement la balise correspondante dans l'onglet HTML.

Voici le code de la partie graphique du fichier `stocks.aspx` après ajout de quelques attributs de style pour contrôler la largeur des éléments (les contrôles serveur sont indiqués en couleurs) :

**stocks.aspx (partie graphique)**

```
<html>
<head>
   <title>Suivi des stocks</title>
   <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
   <SDS:NavBar id="MyNavBar" runat="server" SelectedIndex="1">
   </SDS:NavBar>
   <p></p>
   <p></p>
   <form runat="server">
      <h4>Consultation de l'état du stock par famille de produits</h4>
      <table style="width: 325px" bgcolor="#ffffc0"><tbody>
         <tr>
            <td width="60">Famille</td>
            <td>
               <asp:DropDownList id="ListeFamilles" runat="server"
                width="250px">
               </asp:DropDownList>
            </td>
         </tr>
         </tbody></table>
      <br />
      <asp:DataGrid id="EtatStock" runat="server"></asp:DataGrid>
   </form>
</body>
</html>
```

Nous avons fait la moitié du travail de réalisation de la maquette ; passons maintenant à la page de consultation de l'historique du stock d'un produit donné.

## Maquette de la page de consultation de l'historique du stock d'un produit

Pour réaliser cette page, nous allons partir de l'ébauche de page, enregistrée au paragraphe précédent sous le nom `detailstocks.aspx`, dans laquelle nous allons ajouter :

- un contrôle de type `Label` (étiquette de texte) pour afficher le nom du produit ;
- un contrôle de type `DataGrid` (grille de données) pour afficher l'historique du stock.

Le résultat auquel nous parviendrons lorsque la page sera réalisée est représenté figure 4-3.

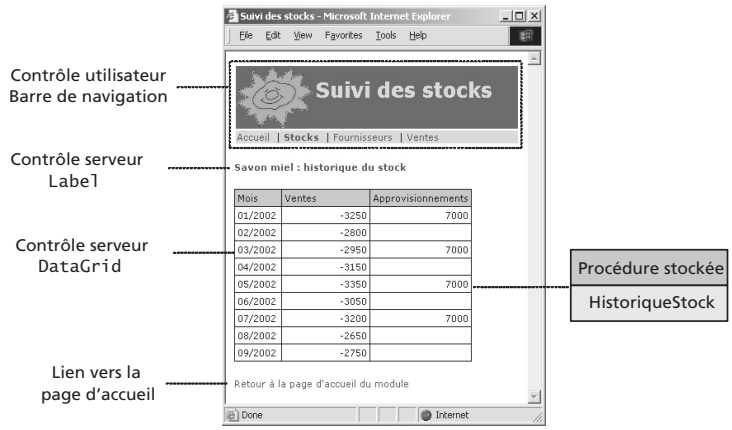



**Figure 4–3** La page de consultation de l'historique du stock pour un produit

Voici les étapes nécessaires à la réalisation de cette page :

1. Ouvrez le fichier `detailstocks.aspx` créé précédemment.
2. Placez-vous dans l'onglet **Design** de Web Matrix.
3. Insérez un contrôle serveur de type `Label`, renommez-le `NomProduit`, puis appliquez-lui le style Heading 4 et faites-le suivre du texte : « historique du stock ».
4. Insérez un contrôle serveur de type `DataGrid` et renommez-le `HistoriqueStock`.
5. Insérez un lien hypertexte « Retour à la page d'accueil du module » pointant vers la page `stocks.aspx`.

Le résultat de la maquette doit ressembler à l'illustration de la figure 4-4.

Voici le code de la partie graphique du fichier `detailstock.aspx` :

```
<html>
<head>
   <title>Suivi des stocks</title>
   <link href="SDS.css" type="text/css" rel="stylesheet" />
</head>
<body>
   <SDS:NavBar id="NavBar" SelectedIndex="1" runat="server">
   </SDS:NavBar>
   <p></p>
   <form runat="server">
      <h4><asp:Label id="NomProduit" runat="server">Label</asp:Label>
      </h4>
      <asp:DataGrid id="HistoriqueStock" runat="server"></asp:DataGrid>
      <p></p>
      <a href="stocks.aspx">Retour à la page d'accueil du module</a>
   </form>
</body>
</html>
```

Nous en avons terminé avec la phase de maquettage… passons maintenant au cœur du problème : l'implémentation du lien entre l'interface et la base de données.

**WEB MATRIX Insérer un lien hypertexte**

Pour insérer un lien hypertexte, il faut choisir Insert HyperLink dans le menu HTML ou, au choix, sélectionner le texte à transformer en lien et utiliser le raccourci clavier Ctrl + K.

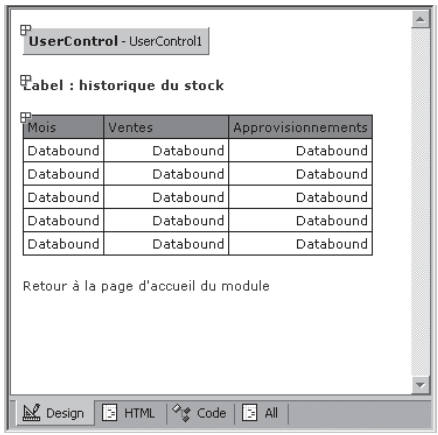



**Figure 4–4** Maquette de la page de consultation de l'historique

# Mise en place des liens entre l'interface et la base de données

À l'image d'un grand nombre de contrôles serveur ASP.NET, `DropDownList` et `DataGrid` offrent la possibilité d'être liés à une source de données : autrement dit, leur contenu peut être automatiquement mis à jour à partir de valeurs contenues dans une base de données, un fichier XML, voire un tableau en mémoire.

Nous allons utiliser cette fonctionnalité pour :

- afficher la liste des familles dans le contrôle `ListeFamilles` à partir des valeurs contenues dans la table `FamilleProduit` ;
- afficher l'état du stock correspondant dans le contrôle `EtatStock` à partir du résultat de l'exécution de la procédure `EtatStock`.

La communication avec la source de données s'effectuera par l'intermédiaire de la bibliothèque ADO.NET, dont nous allons présenter les principales classes.

**Différences entre ADO.NET et ADO**

Même si elles portent des noms proches, les librairies ADO et ADO.NET n'ont que peu de choses en commun : ADO.NET intègre le support natif de SQL Server (ADO utilisait systématiquement OLE-DB), un `DataSet` qui représente un magasin de données en mémoire, pouvant contenir plusieurs tables, être utilisé en mode déconnecté puis transmettre le changement vers la base (alors que le `DataSet` d'ADO ne représentait qu'un seul jeu d'enregistrement et était connecté en permanence à la base de données).

## Présentation de la librairie ADO.NET

ADO.NET est un nom commercial qui désigne un ensemble de classes utilisées pour communiquer avec des sources de données et manipuler des données (figure 4-5) :

- L'espace de nommage `System.Data` fournit des classes utilitaires qui permettent de manipuler des données ; ces classes sont toutes indépendantes de la source de données.
- L'espace de nommage `System.Data.OleDb` fournit les classes nécessaires à la communication avec une source de données OLE-DB.
- L'espace de nommage `System.Data.SqlClient` fournit les classes nécessaires à la communication avec une base de données SQL Server ou MSDE.

**Espace de nommage**

Un espace de nommage (*namespace*, parfois traduit par espace de noms) est un regroupement de classes dans une enveloppe conceptuelle, qui les isole des conflits de noms avec l'extérieur : en pratique, deux classes peuvent porter le même nom si elles sont situées dans deux espaces de nommage différents. Les classes étant généralement regroupées par fonctionnalités, les espaces de nommage ont également un rôle dans l'organisation des bibliothèques de classes.

**OleDb et SqlClient « jumeaux »**

Ces deux espaces de nommage sont en quelque sorte jumeaux, dans la mesure où ils contiennent les mêmes classes préfixées soit par `OleDb`, soit par `Sql` : par exemple, `OleDbConnection` et `SqlConnection`.

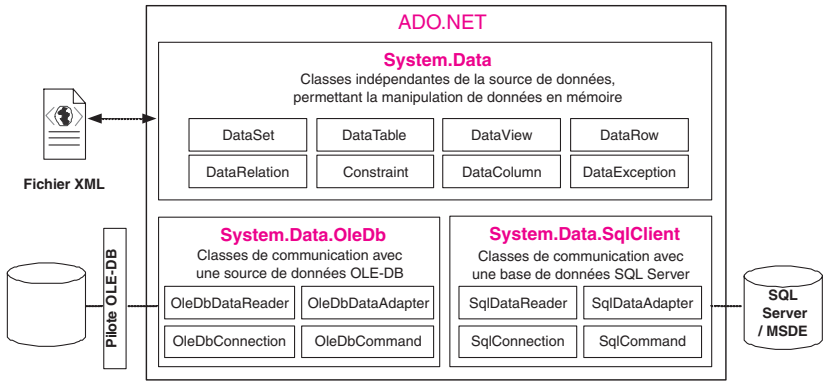



**Figure 4–5** Organisation de la bibliothèque ADO.NET

La première caractéristique notable de l'architecture de cette bibliothèque est la séparation nette entre les classes de communication, dépendantes de la source de données, et les classes de manipulation, indépendantes de la source de données : le but étant de disposer d'une séparation entre la couche présentation et la couche accès aux données, afin de rendre le code plus modulaire et plus facile à faire évoluer.

Le second point notable est la richesse de la bibliothèque `System.Data`, qui dispose de toutes les classes nécessaires à la création d'une véritable petite base de données en mémoire, incluant un schéma relationnel, des contraintes et même des déclencheurs ! L'intérêt principal d'une telle architecture est d'adapter l'accès aux données au caractère déconnecté des applications Web : les classes de manipulation permettent en effet de conserver les données en cache et de transmettre les changements effectués à la base, lors de la connexion suivante (nous aurons l'occasion d'illustrer ce mécanisme dans le chapitre suivant).

À l'heure actuelle, ADO.NET propose deux modes d'accès aux données :

- Utilisation du fournisseur OLE-DB, qui permet d'accéder à toute base de données dotée d'un pilote OLE-DB (classes correspondantes implémentées dans l'espace de nommage `System.Data.OleDb`).
- Utilisation du fournisseur SQL natif, qui permet d'accéder de manière native (autrement dit, plus rapidement et avec accès à l'ensemble des fonctionnalités disponibles) aux bases de données SQL Server ou MSDE (classes correspondantes implémentées dans l'espace de nommage `System.Data.SqlClient`).

Si vous disposez d'une base de données SQL Server ou MSDE, il est fortement recommandé d'utiliser le fournisseur SQL natif, plus performant : c'est ce que nous allons faire dans la section suivante, en implémentant la connexion à la base avec `SqlConnection`.

**À propos d'OLE-DB**

OLE-DB est une spécification d'interface publique pour l'implémentation de pilotes d'accès à des sources de données : c'est un successeur d'ODBC qui ne se limite pas uniquement aux sources de données relationnelles (possibilité d'accéder à un fichier Excel, par exemple).
Pour plus d'informations, voir :
▸ www.microsoft.com/data

**Autres fournisseurs : OBDC, Oracle…**

Il est prévu que la liste des fournisseurs de données compatibles .NET s'étoffe progressivement. Actuellement, deux extensions sont disponibles en téléchargement : un fournisseur ODBC et un fournisseur natif Oracle. Il faut aller sur :
▸ www.microsoft.com/downloads
et chercher « provider ».

## Établissement de la connexion à la base de données

Dans cette section, nous allons voir comment utiliser la bibliothèque ADO.NET pour se connecter à une base de données, à l'aide de la classe `SqlConnection`.

Notre connexion étant destinée à être utilisée plusieurs fois au sein de notre application, nous allons également voir comment la partager, autrement dit, l'implémenter dans un endroit central accessible à tous les éléments de l'application, grâce à l'objet `Session` et au fichier `global.asax`.

## Consulter la documentation des bibliothèques .NET

Pour consulter la liste exhaustive et les caractéristiques des classes contenues dans ADO.NET ou dans les autres bibliothèques .NET, plusieurs moyens sont à notre disposition.

### 1 - Utilisation de Web Matrix

Web Matrix dispose d'un onglet Classes (dans le coin inférieur droit de l'interface) qui permet de consulter la liste des classes disponibles dans les bibliothèques .NET (figure 4-6).

En double-cliquant sur une classe, on fait apparaître le volet de documentation correspondant (figure 4-7).

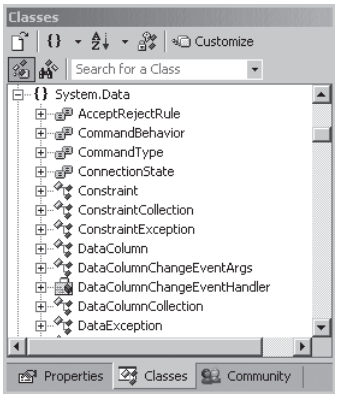



**Figure 4–6** L'onglet Classes de Web Matrix

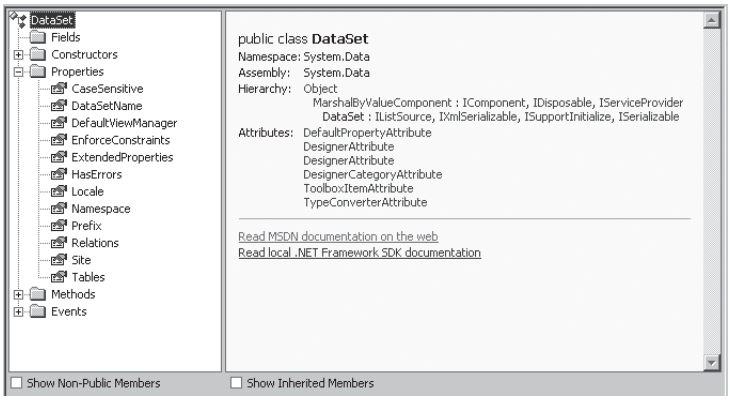



**Figure 4–7** Volet Documentation de Web Matrix

### 2 – Documentation .NET installée sur le poste de travail

Lors de l'installation du .NET Framework SDK sur votre poste de travail (voir chapitre 2), une documentation complète a été installée : elle est accessible via le menu Démarrer (figure 4-8) et détaille de manière exhaustive les classes disponibles (figure 4-9).

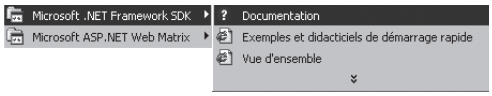



**Figure 4–8** Démarrer la documentation installée en local

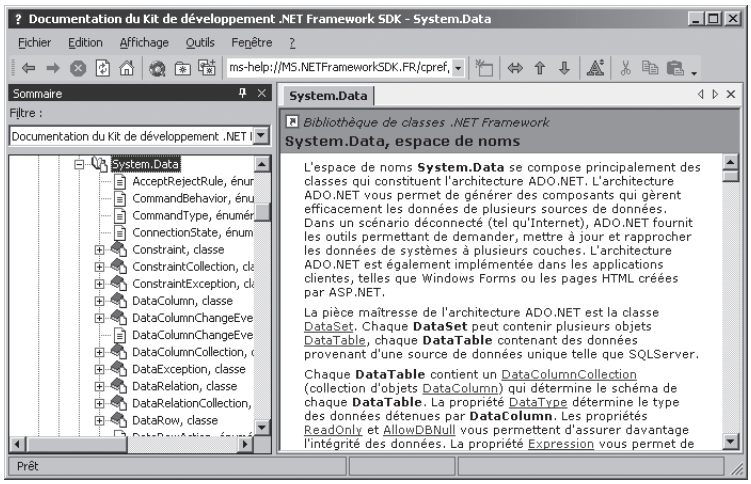



**Figure 4–9** La documentation locale du kit de développement .NET

### 3 – Documentation MSDN en ligne

L'intégralité de la documentation, tenue à jour, est disponible sur le site MSDN (Microsoft Developer Network) à l'adresse :

- http://msdn.microsoft.com

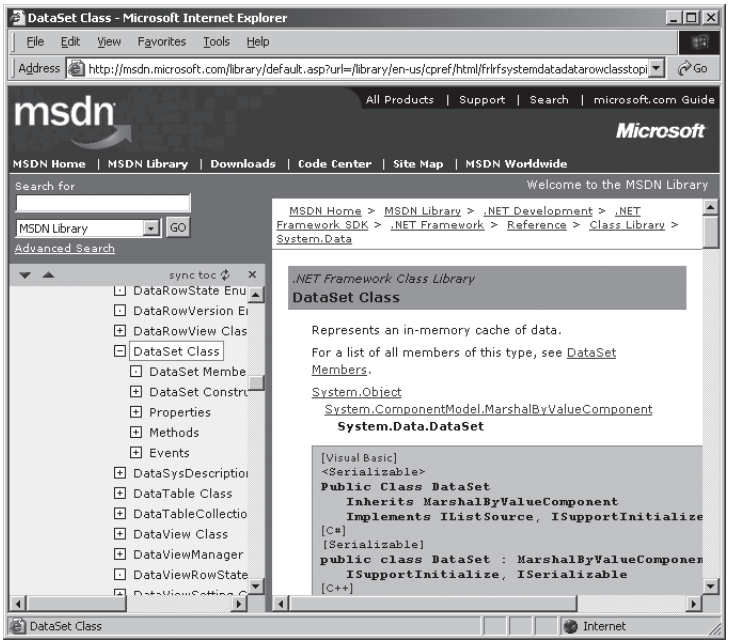



**Figure 4–10** Documentation MSDN en ligne

### 4 –Reflector

Reflector est un utilitaire gratuit qui permet d'examiner, par introspection (*reflection* en anglais, d'où le nom du programme), l'ensemble des assemblages .NET installés sur la machine.

Il dispose d'une interface très ergonomique permettant d'explorer facilement les bibliothèques .NET (avec, en particulier, les liens d'héritage entre classes), d'un moteur de recherche puissant et d'un lien automatique vers les rubriques correspondantes de la documentation MSDN.

Reflector peut être téléchargé gratuitement sur le site de son auteur, Lutz Roeder :

- http://www.aisto.com/roeder/dotnet

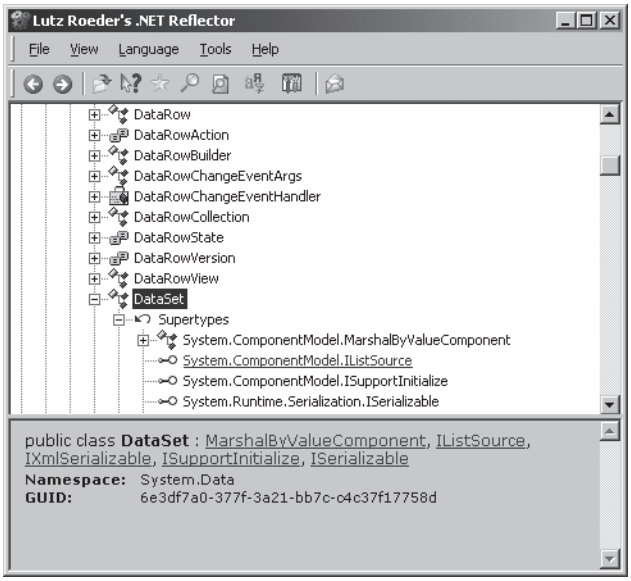



**Figure 4–11** L'utilitaire Reflector

**Chaîne de connexion**

Une chaîne de connexion contient tous les paramètres nécessaires à la connexion à une source de données : nom du serveur, de la base, identifiant et mot de passe de l'utilisateur, et, le cas échéant, des options diverses.

**OleDbConnection**

La classe `OleDbConnection` permet d'établir une connexion à une base de données via un pilote OLE-DB ; elle est équivalente à `SqlConnection` (ces deux classes implémentent l'interface `IDbConnection`).

**Session ASP.NET vs Session ASP ?**

L'objet `Session` d'ASP.NET est une version évoluée de l'équivalent ASP, qui introduit notamment deux innovations appréciables : le support des serveurs multiples (les objets `Session` peuvent être stockés dans une base partagée entre plusieurs serveurs Web, ce qui autorise l'équilibrage de charge tout en supprimant la contrainte de traiter toutes les requêtes d'une même session sur un serveur unique) et la gestion de sessions sans cookies (normalement, les requêtes successives d'un même utilisateur sont détectées grâce à un cookie de session : désormais, il est possible, si besoin est, de gérer les sessions sans ce type de cookie grâce à l'ajout automatique du numéro de session dans chaque chaîne de requête).

## Connexion à la base avec SqlConnection

La connexion à la base est implémentée par une instance `SqlConnection` dont le paramétrage s'effectue par l'intermédiaire d'une chaîne de connexion de la forme :

```
Server=<Server>;Initial Catalog=<Database>;uid=<Login>;password=<Password>
```

où :

- `<Server>` représente le nom du serveur de données (exemple : `(local)\NetSDK`) ;
- `<Database>` représente le nom de la base de données (exemple : `SDS`) ;
- `<Login>` représente l'identifiant utilisé pour se connecter à la base ;
- `<Password>` représente le mot de passe utilisé pour se connecter à la base.

En VB.NET, la syntaxe de connexion à la base est la suivante :

```
Dim myConnection As New SqlConnection
myConnection.ConnectionString="<VotreChaineDeConnexion>"
myConnection.Open()
' Effectuer ici des opérations sur la base
myConnection.Close()
```

Et voici l'équivalent en C# :

```
SqlConnection myConnection = new SqlConnection();
myConnection.ConnectionString="<VotreChaineDeConnexion>"
myConnection.Open();
// Effectuer ici des opérations sur la base
myConnection.Close();
```

Où allons-nous placer ce code de connexion/déconnexion à la base de données ?

Une première option est de le répéter au sein de chaque page : cette solution n'est pas optimale puisqu'elle alourdit le code de l'application, dégrade la performance (nombreuses opérations de connexion/déconnexion) et rend la maintenance difficile en cas de modification des options de connexion.

Une seconde option, préférable, consiste à se connecter au début de la session de l'utilisateur et à conserver la connexion dans un endroit central, accessible à tous les éléments de l'application : c'est cette technique que nous allons exposer dans la section suivante.

## Partage de la connexion à la base avec l'objet Session et le fichier global.asax

Deux éléments vont nous permettre d'implémenter le partage de la connexion à la base de données :

- l'objet `Session`, qui permet de partager des données entre tous les éléments d'une application Web pendant la durée d'une session utilisateur (une instance différente est créée pour chaque session utilisateur) ;

- le fichier `global.asax`, qui permet de stocker des variables globales et d'implémenter des gestionnaires devant être exécutés au début et à la fin d'une session utilisateur, ainsi qu'au démarrage et à l'arrêt d'une application Web.

Voici le processus que nous souhaitons implémenter :

1. Ouverture d'une connexion au démarrage de chaque session utilisateur.
2. Stockage de cette connexion dans le dictionnaire de l'objet `Session` correspondant.
3. Fermeture de la connexion lors de la fin de la session.

Le code correspondant, à implémenter dans le fichier `global.asax` (lequel aura été préalablement créé et placé dans le répertoire racine de l'application), est présenté et commenté ci-après.

**global.asax (Version VB.NET)**

```
<%@Import Namespace="System.Data.SqlClient"%>
<script language="VB" runat="server">

Sub Session_Start(Sender As Object,E As EventArgs )
   Dim myConnection As SqlConnection
   myConnection = new SqlConnection()
   myConnection.ConnectionString="<VotreChaineDeConnexion>"
   myConnection.Open()
   Session("myConnection") = myConnection
End Sub

Sub Session_End(Sender As Object,E As EventArgs )
   Dim myConnection As SqlConnection
   myConnection = CType(Session("myConnection"),SqlConnection)
   myConnection.Close()
End Sub
</script>
```

L'instruction `<%@ Import...>` permet d'inclure une référence à un espace de nommage, ce qui permet ensuite de référencer les classes de cet espace par leur nom court : par exemple `SqlConnection`, au lieu de `System.Data.SqlClient.SqlConnection`.

Dans ce gestionnaire exécuté au début de chaque session, on crée et on ouvre la connexion à la base, puis on stocke l'objet correspondant dans le dictionnaire de l'objet `Session` (remplacer `<VotreChaineDeConnexion>` par votre véritable chaîne de connexion).

Dans ce gestionnaire exécuté à la fin de chaque session, on ferme la connexion à la base.

Voici le code correspondant en C# :

**global.asax (Version C#)**

```
<%@Import Namespace = "System.Data.SqlClient"%>
<script language="C#" runat="server">

void Session_Start(Object sender, EventArgs E)
{
   SqlConnection myConnection = new SqlConnection();
   myConnection.ConnectionString="<VotreChaineDeConnexion>"
   myConnection.Open();
   Session["myConnection"]=myConnection;
}
```

**global.asax vs global.asa ?**

Le fichier `global.asax` est l'équivalent ASP.NET du fichier `global.asa`. À la différence de son prédécesseur, le code qu'il contient peut être rédigé dans n'importe quel langage compatible .NET (VB.NET, C#, etc.). L'extension `.asax` est utilisée afin de ne pas interférer avec les anciens fichiers `.asa` (autrement dit, les fichiers `global.asax` et `global.asa` peuvent cohabiter dans la même application). À ce sujet, il faut veiller à bien placer le fichier `global.asax` à la racine de l'application pour qu'il soit correctement pris en compte.

**ALTERNATIVE Utiliser le fichier web.config**

Une alternative courante consiste à stocker la chaîne de connexion dans la section `<appSettings>` du fichier `web.config`, ce qui la rend accessible via les membres statiques de la classe `ConfigurationSettings`.

**DANS UN CAS RÉEL** **Protection du mot de passe**

Dans le cas d'une application réelle, le mot de passe ne serait vraisemblablement pas laissé en clair dans un fichier texte : on utiliserait par exemple le mot de passe fourni par un utilisateur lors de la connexion à l'application.

```
void Session_End(Object sender, EventArgs E)
{
   SqlConnection myConnection = (SqlConnection)Session["myConnection"];
   myConnection.Close();
}
</script>
```

**Performance et consommation de ressources**

En ce qui concerne le partage de connexion, on peut se demander s'il vaut mieux conserver une connexion ouverte pendant toute la durée de la session (option choisie dans notre cas, plus rapide à l'exécution mais plus consommatrice de ressources serveur) ou partager uniquement la chaîne de connexion et ouvrir/fermer une connexion à chaque exécution de page (option plus lente mais moins consommatrice de ressources) : en pratique, les fournisseurs .NET étant dotés d'un mécanisme de mise en cache de la connexion (*connection pooling*), les deux options sont équivalentes !

Maintenant que nous sommes connectés à la base de données, passons à l'établissement du lien entre cette connexion et les contrôles.

**Exécution en mode Debug**

Si le caractère compilé des pages ASP.NET élimine un grand nombre d'erreurs dès la phase de développement, il est néanmoins toujours possible qu'il se produise des erreurs à l'exécution : on parle alors d'exceptions.

Par exemple, si la page ASP.NET n'est pas capable de se connecter à la base de données SQL Server (serveur indisponible, nom d'utilisateur incorrect, etc.), une exception de type `SqlException` surviendra (voir figure 4-12).

Pour savoir quelle est la ligne de code qui a provoqué l'exception, il est nécessaire que la page soit exécutée en mode Debug, lequel peut être activé de deux manières différentes :

- par l'ajout d'une directive `Debug="true"` dans la directive `<% Page %>` ;
- par l'ajout d'une section `<compilation debug="true">` dans le fichier `web.config`.

Nous avons déjà eu l'occasion de parler, au chapitre précédent, de la directive `<%@ Page ...%>` qui, placée en haut du fichier `.aspx`, permet notamment de spécifier le langage utilisé dans la page. Cette même directive permet d'activer le mode Debug pour la page :

```
<% @ Page Language="C#" Debug="true" %>
```

Si l'on souhaite activer le mode Debug pour l'ensemble des pages de l'application, il est plus simple de créer un fichier de configuration de l'application nommé `web.config` et placé dans le répertoire racine de l'application : ce fichier permet de spécifier les paramètres d'exécution de l'application (nous aurons l'occasion d'en reparler au chapitre 9).

**Fichier web.config**

```
<configuration>
    <system.web>
       <compilation debug="true"/>
    </system.web>
</configuration>
```

Notez que le mode Debug est déconseillé pour des applications en production : il ralentit considérablement l'exécution et, de plus, rend le code source en partie visible par les utilisateurs en cas d'erreur.

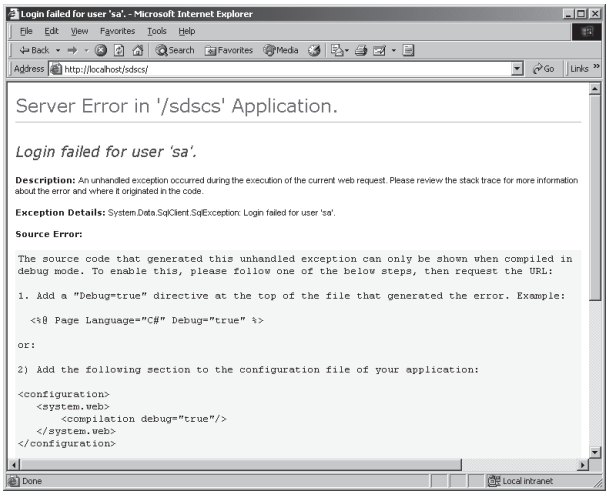



**Figure 4–12** Écran indiquant qu'une exception est survenue

## Liaison du contrôle DropDownList à la table FamilleProduit

Comme nous l'avons indiqué plus haut, le contrôle serveur `DropDownList` offre la possibilité d'être automatiquement rempli à partir de valeurs contenues dans une base de données.

En pratique, ce lien s'effectue à l'aide de trois propriétés du contrôle :

- la propriété `DataSource`, utilisée pour spécifier la source de données (dans notre cas, la table `FamilleProduit`) ;
- la propriété `DataTextField`, utilisée pour spécifier le nom du champ à afficher dans la liste (dans notre cas, le champ `NomFamille`) ;
- la propriété `DataValueField`, utilisée pour spécifier la valeur à associer aux éléments de la liste (dans notre cas, le champ `ID_FamilleProduit`).

Pour établir le lien entre la table de données et la liste déroulante, deux options sont possibles :

- utiliser des classes `SqlCommand` et `SqlDataReader` ;
- utiliser des classes `SqlDataAdapter`, `DataTable` et `DataView`.

L'option avec utilisation de `SqlDataReader` est, a priori, la plus simple et la plus performante ; quant à la seconde option, elle est fréquemment présentée dans la documentation .NET et permet de présenter des classes importantes comme `SqlDataAdapter` et `DataTable`.

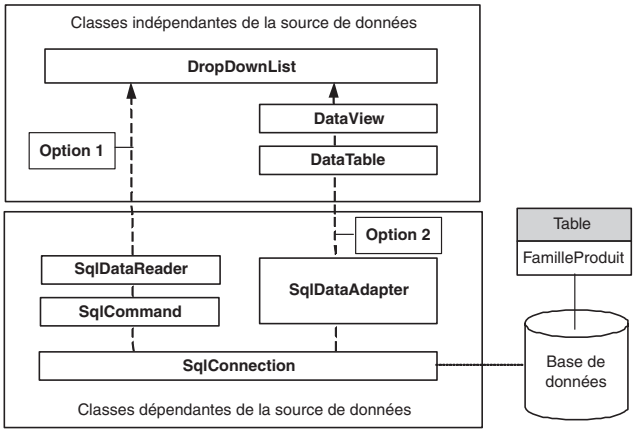



**Figure 4–13** Options possibles pour le lien avec la base de données

Nous allons donc implémenter deux versions d'une fonction permettant d'effectuer le chargement de la liste des familles, que nous nommerons `ChargerListeFamilles`. Puis, nous intégrerons cette fonction dans notre page `stocks.aspx`.

**Jeu de résultat**

Désigne un ensemble de données organisées en lignes et en colonnes (contenu d'une table, d'une vue, résultat de l'exécution d'une procédure stockée…).

**À propos de SqlDataReader**

Un objet de type `SqlDataReader` permet de balayer un jeu de résultat, en lecture seule, de la première vers la dernière ligne : l'avantage principal est la performance (seule la ligne active du jeu de résultat est conservée en mémoire). En revanche, les fonctionnalités de manipulations de données sont beaucoup plus réduites qu'avec un objet de type `DataTable` (décrit plus loin).

## Alternative 1 - Utilisation de SqlCommand et SqlDataReader

Cette première option utilise trois classes extraites de l'espace de nommage `System.Data.SqlClient` :

- La classe `SqlConnection`, déjà décrite précédemment, implémente la connexion à la base de données.
- La classe `SqlCommand` permet de spécifier à quel objet de la base de données on s'intéresse, en l'occurrence la table `FamilleProduit`.
- La classe `SqlDataReader` permet de manipuler les données, en l'occurrence les parcourir en lecture seule.

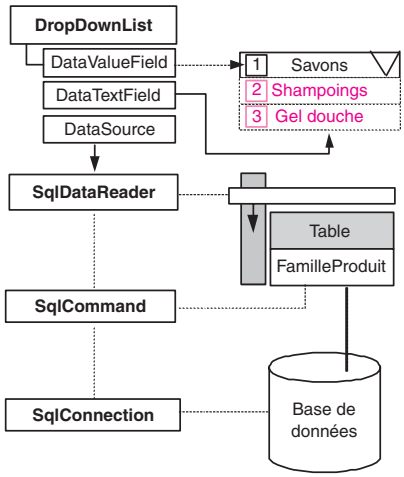



**Figure 4–14** Accès à la base en utilisant SqlDataReader

Voici la version correspondante de la fonction `ChargerListeFamilles` :

**Stocks.aspx - Fonction ChargerListeFamilles (version VB.NET/SqlDataReader)**

```
Sub ChargerListeFamilles()
```

Déclaration des variables

```
    Dim myConnection As SqlConnection
    Dim myCommand As SqlCommand
    Dim myReader  As SqlDataReader
    Dim SQL As String
```

Récupération de la connexion (stockée dans l'objet `Session`, grâce au fichier `global.asax`)

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

Allocation de l'objet `SqlCommand` (« connecte-toi à telle base et intéresse-toi à telles données »)

```
    SQL = "SELECT * FROM FamilleProduit"
    myCommand = new SqlCommand(SQL,myConnection)
```

Allocation de l'objet `SqlDataReader` par l'intermédiaire de l'objet `SqlCommand` (« fournis-moi un objet capable de parcourir les données »)

```
    myReader = myCommand.ExecuteReader()
```

```
    ListeFamilles.DataSource      = myReader
    ListeFamilles.DataValueField = "ID_FamilleProduit"
    ListeFamilles.DataTextField  = "NomFamille"

    ListeFamilles.DataBind()

    myReader.Close()

End Sub
```

Paramétrage de l'objet `ListeFamilles`

Chargement effectif de la liste (sans cette ligne, la liste resterait vide !)

Fermeture de l'objet `SqlDataReader` (indispensable, car `myReader` a été ouvert par `ExecuteReader`…)

Et voici la version équivalente en C# :

**Stocks.aspx – Fonction ChargerListeFamilles (version C#/SqlDataReader)**

```
void ChargerFamillesProduits()
{
   SqlCommand    myCommand;
   SqlDataReader myReader;
   SqlConnection myConnection = (SqlConnection)Session["myConnection"];

   string 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();
}
```

Avant de tester le résultat du chargement de la liste des familles de produits, nous présentons la deuxième option possible pour accéder aux données.

## Alternative 2 - Utilisation de SqlDataAdapter, DataTable et DataView

Cette deuxième option utilise quatre classes :

- La classe `SqlConnection`, déjà décrite précédemment, implémente la connexion à la base de données.
- La classe `SqlDataAdapter` permet de gérer la communication avec un objet de la base de données de manière bidirectionnnelle (lecture, mise à jour, suppression de données).
- La classe `DataTable` représente une table en mémoire dont il est possible de manipuler les données, en lecture et en écriture.
- La classe `DataView` représente une vue d'une table, autrement dit, un sous-ensemble énumérable des lignes de la table.

Les deux premières classes font partie de l'espace de nommage `System.Data.SqlClient` (elles sont donc dépendantes de la base de données) tandis que les deux dernières sont extraites de `System.Data` (elles sont donc indépendantes de la base de données).

**ASP.NET Vers quoi peut pointer la propriété DataSource ?**

La propriété DataSource d'un contrôle lié aux données ne doit pas nécessairement pointer vers une base de données relationnelle (via DataView ou DataReader) : il est également possible de lier un contrôle serveur à un tableau de données en mémoire (ArrayList, HashTable) ou au contenu d'un fichier XML (via XmlNodeList). La seule contrainte est que l'objet pointé permette d'énumérer des données – en termes techniques, qu'il implémente une des interfaces suivantes : ICollection, IEnumerable ou IListSource. Dans le cas d'un contrôle lié à une base de données, le choix le plus courant est l'utilisation d'un objet de type DataView, qui représente un jeu de données issues d'une table.

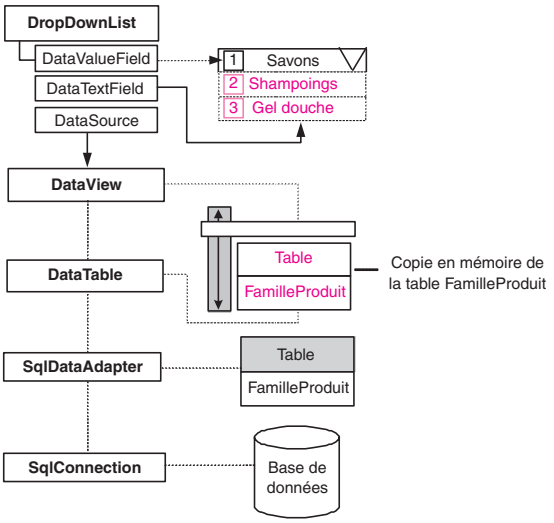



**Figure 4–15** Accès à la base en utilisant SqlDataAdapter

Comme on le voit sur la figure 4-15, cette solution constitue en quelque sorte « la grosse artillerie » pour remplir une liste déroulante :

- Une copie de la table FamilleProduit est conservée en mémoire (gérée par DataTable), ce qui consomme inutilement des ressources.
- La classe SqlDataAdapter est utilisée pour remplir cette table en mémoire ; néanmoins, dans notre cas, elle est sous-utilisée car elle pourrait également servir à synchroniser les changements effectués sur la version « en mémoire » vers la base de données.

L'intérêt de présenter cette solution est donc plus didactique que technique !

Voici la version correspondante de la fonction ChargerListeFamilles :

**Stocks.aspx – Fonction ChargerListeFamilles (version VB.NET/SqlDataAdapter)**

| | |
|---|---|
| | `Sub ChargerListeFamilles()` |
| Déclaration des variables utilisées | `Dim myConnection As SqlConnection`<br>`Dim myAdapter As SqlDataAdapter`<br>`Dim myDataTable As DataTable`<br>`Dim SQL As String` |
| Récupération de la connexion (stockée dans l'objet Session) | `myConnection = CType(Session("myConnection"),SqlConnection)` |
| Allocation de l'objet SqlDataAdapter (lié à la table FamilleProduit) | `SQL = "SELECT * FROM FamilleProduit"`<br>`myAdapter = new SqlDataAdapter(SQL,myConnection)` |
| Allocation de l'objet DataTable | `myDataTable = new DataTable()` |
| Remplissage de l'objet DataTable | `myAdapter.Fill(myDataTable)` |
| Initialisation du contrôle ListeFamilles (l'appel à DataBind réalise le chargement de la liste) | `ListeFamilles.DataSource = myDataTable.DefaultView`<br>`ListeFamilles.DataValueField = "ID_FamilleProduit"`<br>`ListeFamilles.DataTextField = "NomFamille"`<br>`ListeFamilles.DataBind()`<br>`End Sub` |

Voici le code correspondant en C# :

**Stocks.aspx – Fonction ChargerListeFamilles (version C#/SqlDataAdapter)**

```
void ChargerListeFamilles()
{
   SqlConnection myConnection = (SqlConnection)Session["myConnection"];
   SqlDataAdapter   myAdapter;
   DataTable        myDataTable;

   string strSQL = "SELECT * FROM FamilleProduit";

   myAdapter  = new SqlDataAdapter(strSQL,myConnection);
   myDataTable = new DataTable();
   myAdapter.Fill(myDataTable);

   ListeFamilles.DataSource = myDataTable.DefaultView;
   ListeFamilles.DataValueField = "ID_FamilleProduit";
   ListeFamilles.DataTextField = "NomFamille";
   ListeFamilles.DataBind();
}
```

## Tester le remplissage de la liste des familles de produits

Avant de tester notre page, il faut réaliser deux opérations supplémentaires :

- ajout des directives `<%@ Import %>` ;
- implémentation de l'appel de la fonction `ChargerListeFamilles` depuis `Page_Load`.

Dans notre cas, voici les directives `<%@ Import %>` à spécifier :

**Stocks.aspx (début du fichier)**

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

Enfin, il ne reste plus qu'à rajouter un appel à la fonction `ChargerListeFamilles` dans le gestionnaire `Page_Load` de notre page :

**Stocks.aspx – Fonction Page_Load (version VB.NET)**

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

**Stocks.aspx – Fonction Page_Load (version C#)**

```
void Page_Load(Object sender, EventArgs e)
{
   ChargerListeFamilles();
}
```

**RAPPEL Les directives <%@ Import %>**

Les directives `<%@ Import ...%>` permettent de spécifier les espaces de nommage pour lesquels on souhaite pouvoir faire référence aux classes contenues par leur nom court.

Si vous avez choisi l'alternative 1 pour le chargement de la liste des familles (`SqlDataReader`), seule la référence à `System.Data.SqlClient` est nécessaire.

**ATTENTION Position des directives Import**

Les directives `<%@ Import ...%>` doivent être placées hors du bloc `<script runat="server">`; en général, elles sont placées en haut du fichier. Avec Web Matrix, positionnez-vous dans l'onglet All pour saisir ces directives (si vous êtes dans l'onglet Code, les directives seront insérées dans le bloc `<script>` !).

**À propos de Page_Load**

Page_Load est une fonction appelée au chargement de la page. Nous aurons l'occasion de présenter plus en détail le mécanisme des événements au sein d'une page ASP.NET à la fin de ce chapitre.

**Espaces de nommage importés implicitement**

Les espaces de nommage suivants sont implicitement importés dans les pages ASP.NET :

- System
- System.Collections
- System.Collections.Specialized
- System.Web
- System.Web.UI
- System.Web.UI.WebControls
- System.Web.UI.HtmlControls
- System.Web.Caching
- System.Web.Security
- System.Web.SessionState
- System.Text
- System.Text.Configuration
- System.Text.RegularExpressions

Voilà ! Si tout se passe bien, vous devez maintenant voir apparaître la liste des familles de produits remplie lorsque vous exécutez la page (voir figure 4-16).

**Figure 4–16** Remplissage de la liste des familles de produits

Nous avons réalisé la première moitié de la page. Passons maintenant à la seconde : l'implémentation du contrôle `DataGrid` destiné à afficher l'état des stocks.

## Utilisation de DataGrid pour afficher l'état des stocks

La liaison entre le contrôle `EtatStock` de type `DataGrid` et la procédure stockée `EtatStock` va s'effectuer de manière similaire à ce que nous avons réalisé dans la section précédente.

Rappelons que deux techniques sont à notre disposition :

- utiliser `SqlCommand` et `SqlDataReader` ;
- utiliser `SqlDataAdapter`, `DataTable` et `DataView`.

Nous allons ici opter pour l'utilisation de `SqlDataReader`, la technique la plus performante et la mieux adaptée à nos besoins. Encore une fois, nous allons implémenter le paramétrage dans une fonction distincte, que nous nommerons `AfficherStockProduits`, profitant ainsi des possibilités de structuration du code offertes par ASP.NET.

La seule particularité supplémentaire est ici l'emploi d'une procédure stockée (ce que nous spécifions grâce à la propriété `CommandType` de `SqlCommand`) qui accepte en entrée un paramètre de type entier égal au numéro de la famille sélectionnée dans `ListeFamilles`.

**Stocks.aspx – Fonction AfficherStocksProduits (version VB.NET)**

Déclaration des variables

```
Sub AfficherStocksProduits()
   Dim myCommand As SqlCommand
   Dim myReader  As SqlDataReader
   Dim myConnection As SqlConnection
   Dim FamilleID As String
```

```
    myConnection = CType(Session("myConnection"),SqlConnection)
    FamilleID    = ListeFamilles.SelectedItem.Value

    myCommand = new SqlCommand("EtatStock",myConnection)
    myCommand.CommandType = CommandType.StoredProcedure
    myCommand.Parameters.Add("@FamilleID",SqlDbType.Int).Value = FamilleID
    myReader = myCommand.ExecuteReader()

    EtatStock.DataSource = myReader
    EtatStock.DataBind()

    myReader.Close()

End Sub
```

- Récupération de la connexion ainsi que du numéro de la famille sélectionnée
- Construction de l'objet `SqlCommand` associé à la procédure `EtatStock` et spécification du paramètre d'entrée, égal au numéro de la famille sélectionnée
- Paramétrage de la grille de données
- Fermeture de l'objet `SqlDataReader` (à ne pas oublier !)

**Stocks.aspx – Fonction AfficherStocksProduits (version C#)**

```
void AfficherStocksProduits()
{
   SqlCommand       myCommand;
   SqlDataReader    myReader;
   SqlConnection    myConnection = (SqlConnection)Session["myConnection"];
   string FamilleID = ListeFamilles.SelectedItem.Value;

   myCommand = new SqlCommand("EtatStock",myConnection);
   myCommand.CommandType = CommandType.StoredProcedure;
   myCommand.Parameters.Add("@FamilleID",SqlDbType.Int).Value = FamilleID;
   myReader = myCommand.ExecuteReader();

   EtatStock.DataSource = myReader;
   EtatStock.DataBind();

   myReader.Close();
}
```

**ALTERNATIVE Syntaxe in-line pour la procédure stockée**

Au lieu de recourir aux paramètres explicites, il est possible d'utiliser la syntaxe dite en ligne (*in-line*) pour l'appel de la procédure stockée :

```
string SQL = "EtatStock '"+FamilleID+"'"
myCommand = new SqlCommand(SQL,
            ➥ myConnection)
```

Avant de tester notre page, il ne reste plus qu'à ajouter un appel à la fonction `AfficherStocksProduits` dans le gestionnaire `Page_Load` :

**Stocks.aspx – Fonction Page_Load (version VB.NET)**

```
Sub Page_Load(sender As Object, e As EventArgs)
   ChargerListeFamilles()
   AfficherStocksProduits()
End Sub
```

**Stocks.aspx – Fonction Page_Load (version C#)**

```
void Page_Load(Object sender, EventArgs e)
{
   ChargerListeFamilles();
   AfficherStocksProduits();
}
```

**Figure 4–17** La page de suivi des stocks avec le contrôle `DataGrid`

Le résultat de l'exécution de la page est présenté sur la figure 4-17 : on peut apprécier le peu de lignes de code qui ont été nécessaires pour générer un tableau HTML complet !

Néanmoins, l'aspect du tableau laisse encore un peu à désirer ; de plus, il ne semble pas indispensable d'afficher les colonnes ID_Produit et ID_FamilleProduit…

Nous allons voir dans la section suivante comment personnaliser le contrôle `DataGrid` pour qu'il soit mieux adapté à nos besoins.

## Personnalisation du contrôle DataGrid

Le contrôle `DataGrid` dispose de nombreuses possibilités de paramétrage, notamment :

- le contrôle des colonnes à afficher et celui du format de ces colonnes (alignement, police) ;
- le contrôle des couleurs (en-tête, pied de tableau, lignes) ;
- la gestion de la pagination (navigation entre les pages).

Tous ces paramétrages peuvent être effectués via une interface graphique :

1. Sélectionnez le contrôle `EtatStock` dans l'onglet Design de Web Matrix.
2. Dans la feuille de propriétés associée, cliquez sur le lien PropertyBuilder (figure 4-18).
3. Une boîte de dialogue permettant le paramétrage de `EtatStock` apparaît (figure 4-19).

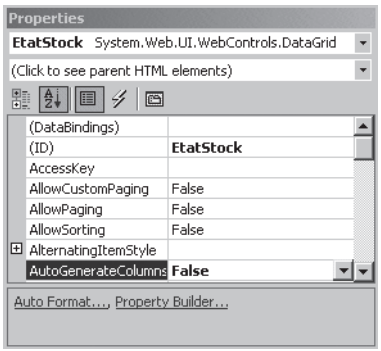



**Figure 4–18** Feuille de propriétés EtatStock

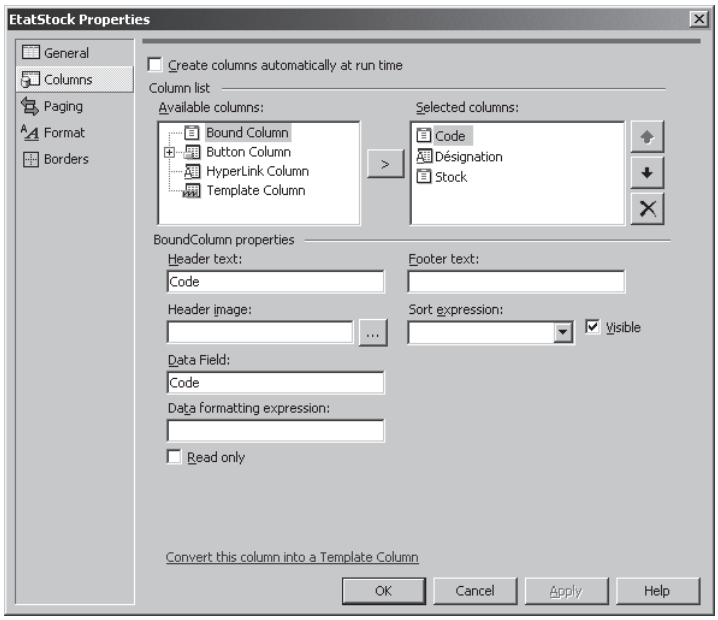



**Figure 4–19** Paramétrage du contrôle EtatStock

Par défaut, les colonnes du `DataGrid` sont générées dynamiquement à partir des colonnes présentes dans la source de données. Dans notre cas, nous souhaitons n'afficher que les colonnes Code, Désignation et Stocks, à l'exclusion des colonnes clés (ID_Produit et ID_FamilleProduit). En outre, les éléments de la colonne Désignation doivent agir comme des liens hypertextes qui permettent d'accéder à l'historique du stock.

Pour cela, nous allons désactiver la fonctionnalité de génération automatique des colonnes (propriété `AutoGenerateColumns`) et spécifier une liste personnalisée de colonnes :

- colonne Code de type `BoundColumn` ;
- colonne Désignation de type `HyperLinkColumn` ;
- colonne Stocks de type `BoundColumn`.

Voici la marche à suivre pour spécifier notre propre liste de colonnes :

1. Faites apparaître la boîte de dialogue de paramétrage de `EtatStock` (voir précédemment).
2. Cliquez sur Columns afin de faire apparaître la feuille de propriétés relative aux colonnes.
3. Désactivez l'option Create columns automatically at run time.
4. Créez une nouvelle colonne de type `BoundColumn` ayant les caractéristiques suivantes :

| | | |
|---|---|---|
| **Header Text** | Code | Le texte de l'en-tête de la colonne |
| **Data Field** | Code | Le champ correspondant de la base |

5. Créez une nouvelle colonne de type `HyperLinkColumn` ayant les caractéristiques suivantes :

| | | |
|---|---|---|
| **Header Text** | Désignation | Le texte de l'en-tête de la colonne |
| **Text Field** | Code | Le champ correspondant de la base |
| **Url Field** | ID_Produit | Le champ à rendre disponible dans l'URL |
| **Url Format String** | DetailsStocks.aspx?id={0} | L'URL associée ({0} désignant Url Field) |

6. Créez une nouvelle colonne de type `BoundColumn` ayant les caractéristiques suivantes :

| | | |
|---|---|---|
| **Header Text** | Stocks | Le texte de l'en-tête de la colonne |
| **Data Field** | Stocks | Le champ correspondant de la base |

7. Cliquez sur OK pour enregistrer les modifications.

**Les types de colonnes gérés par DataGrid**

Une grille de données (contrôle `DataGrid`) peut contenir des colonnes de type `BoundColumn` (colonne simple liée à un champ de la base), `HyperLinkColumn` (colonne contenant des liens hypertextes), `ButtonColumn` (colonne contenant des boutons d'actions), `TemplateColumn` (colonne dont le format est fixé par l'utilisateur) et `EditCommandColumn` (colonne permettant d'éditer les valeurs de la ligne).

Enfin, nous allons personnaliser l'aspect graphique de notre grille de données (couleur de l'en-tête, espacement entre les cellules, largeur des colonnes) :

1. Faites apparaître la boîte de dialogue de paramétrage de `EtatStock` (voir plus haut).
2. Cliquez sur **Format** afin de faire apparaître la feuille de propriétés relative au formatage.
3. Sélectionnez l'élément `Header` dans la liste `Objects` et spécifiez :

| | |
|---|---|
| Back Color | #AAAADD |

4. Sélectionnez l'élément `Columns` dans la liste `Objets` et spécifiez les largeurs suivantes :

| | |
|---|---|
| Colonne 0 (Code) | 50 |
| Colonne 1 (Désignation) | 100 |
| Colonne 2 (Stock) | 50 |

5. Pour la colonne 2 (Stocks), sélectionnez l'élément fils `Items` et spécifiez :

| | |
|---|---|
| Horizontal alignement | Right |

6. Cliquez sur **Borders** afin de faire apparaître la feuille de propriétés relative aux bordures.
7. Spécifiez les caractéristiques suivantes pour les bordures :

| | |
|---|---|
| Cell padding | 3 |
| Grid lines | Both |
| Border color | Black |
| Border width | 1 px |

8. Cliquez sur OK pour enregistrer les modifications.

Jetez un coup d'œil à l'onglet HTML : vous constaterez que la partie relative au contrôle `EtatStock` a été modifiée en conséquence.

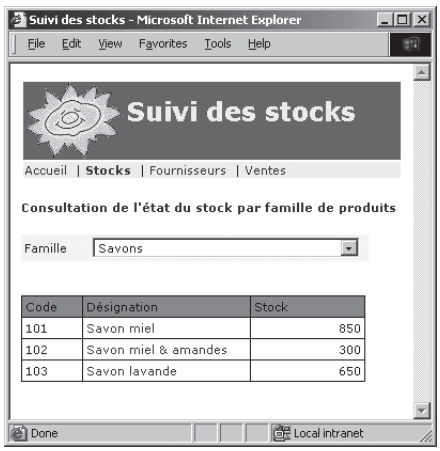



**Figure 4–20** Page de suivi des stocks après paramétrage

**stocks.aspx (contenu HTML) – Après paramétrage du DataGrid**

```
<asp:DataGrid id="EtatStock" runat="server" BorderColor="black"
   CellPadding="3" BorderWidth="1" AutoGenerateColumns="false"
   HeaderStyle-BackColor="#aaaadd" CellSpacing="0" GridLines="Both">

  <Columns>
    <asp:BoundColumn HeaderText="Code" DataField="Code">
        <HeaderStyle width="50px"></HeaderStyle>
    </asp:BoundColumn>
    <asp:HyperLinkColumn
          HeaderText="Désignation"
          DataNavigateUrlField="ID_Produit"
          DataNavigateUrlFormatString="DetailStocks.aspx?id={0}"
          DataTextField="Designation">
```

```
        <HeaderStyle width="150px"></HeaderStyle>
    </asp:HyperLinkColumn>
    <asp:BoundColumn HeaderText="Stock"
                     ItemStyle-HorizontalAlign="right"
                     DataField="Stock">
        <HeaderStyle width="100px"></HeaderStyle>
    </asp:BoundColumn>
  </Columns>
</asp:DataGrid>
```

Exécutez la page pour admirer le résultat (voir figure 4-20) !

Il est maintenant temps de réaliser la page `detailstocks.aspx` vers laquelle pointent les liens hypertextes de la colonne Désignation.

## Implémentation de la page de consultation de l'historique

La page `detailstocks.aspx` affiche l'historique des variations de stocks (approvisionnements, ventes) d'un produit donné.

Nous avons déjà réalisé la maquette de cette page au début du chapitre ; l'implémentation doit réaliser les étapes suivantes :

1. Récupération du numéro de produit sur la ligne de requête.
2. Affichage du nom du produit concerné dans le contrôle `Label`.
3. Paramétrage du contrôle `DataGrid`, lié à la procédure `HistoriqueStock`.

La récupération du paramètre sur la ligne de commande s'effectuera par l'intermédiaire de l'objet global `Request`, qui contient toutes les informations relatives à la requête HTTP correspondant à la page : paramètres de la chaîne de requête, informations sur l'utilisateur (adresse IP, type de navigateur, cookies…) et valeurs des champs de formulaire.

Pour la récupération des informations dans la base, nous allons utiliser la procédure `HistoriqueStock`, qui permet à la fois de récupérer le nom du produit et l'historique du stock pour ce produit, à condition de passer le numéro du produit en paramètre.

Voici le code correspondant :

**DetailStock.aspx (Version VB.NET)**

```
Sub Page_Load(sender As Object,e As EventArgs)
    Dim myConnection As SqlConnection
    Dim myReader  As SqlDataReader
    Dim myCommand As SqlCommand
    myConnection = CType(Session("myConnection"),SqlConnection)

    myCommand = new SqlCommand("HistoriqueStock",myConnection)
    myCommand.CommandType = CommandType.StoredProcedure
    myCommand.Parameters.Add("@ProduitID",SqlDbType.Int).
    myCommand.Parameters("@ProduitID").Value = Request.Params("id")
```

Déclaration des variables

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

Définition de la commande associée à la procédure stockée (le numéro de produit, récupéré sur la ligne de commande, est passé en paramètre de la procédure)

**Objet Request : version ASP.NET vs ASP**

L'objet `Request` d'ASP.NET est une version améliorée de son correspondant ASP : notons, entre autres, que tous les paramètres sont accessibles via une collection nommée `Params`, qui regroupe les informations de la chaîne de requête, les variables serveur, les cookies, etc.

**Objets globaux en ASP.NET**

Plusieurs objets globaux sont accessibles depuis une page ASP.NET : aux objets `Request`, `Response`, `Application`, `Server` et `Session`, qui correspondent à leurs prédécesseurs ASP, s'ajoutent deux nouveaux objets : `Page` et `Cache`.

Récupération du nom du produit (un premier SqlDataReader est mis en œuvre ; la méthode Read permet de se positionner sur le premier enregistrement ; 4 est l'indice de la colonne contenant le nom du produit)

```
    myReader = myCommand.ExecuteReader()
    myReader.Read()
    NomProduit.Text = myReader.GetString(4)
    myReader.Close()
```

Paramétrage de la grille de données (on réutilise la variable myReader pour la faire pointer vers un nouveau SqlDataReader ; l'ancien objet sera « désalloué » automatiquement par le garbage collector du CLR)

```
    myReader = myCommand.ExecuteReader()
    HistoriqueStocks.DataSource = myReader
    HistoriqueStocks.DataBind()
    myReader.Close()
End Sub
```

**DetailStock.aspx (Version C#)**

```
void Page_Load(Object sender, EventArgs e)
{
    SqlDataReader myReader;
    SqlCommand    myCommand;
    SqlConnection myConnection;

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

    myCommand = new SqlCommand("HistoriqueStock",myConnection);
    myCommand.CommandType = CommandType.StoredProcedure;
    myCommand.Parameters.Add("@ProduitID",SqlDbType.Int);
    myCommand.Parameters["@ProduitID"].Value = Request.Params["id"];

    myReader = myCommand.ExecuteReader();
    myReader.Read();
    NomProduit.Text = myReader.GetString(4);
    myReader.Close();

    myReader = myCommand.ExecuteReader();
    HistoriqueStocks.DataSource = myReader;
    HistoriqueStocks.DataBind();
    myReader.Close();
}
```

Pour finir, paramétrons l'aspect graphique du contrôle DataGrid, comme nous l'avons fait déjà fait précédemment :

1. Faites apparaître la boîte de dialogue de paramétrage associée à HistoriqueStocks.
2. Cliquez sur Columns afin de faire apparaître la feuille de propriétés relative aux colonnes.
3. Désactivez l'option Create columns automatically at run time.
4. Créez trois colonnes de type Bound Column, ayant les caractéristiques suivantes :

| Header Text | Data Field |
|---|---|
| Mois | DateMouvement |
| Ventes | Ventes |
| Approvisionnements | Approvisionnements |

5 Spécifiez les mêmes paramétrages de couleurs et de bordure que pour le contrôle `EtatStock` de la page `stocks.aspx`.

6 Cliquez sur OK pour enregistrer les changements.

Le résultat de l'exécution de la page est représenté à la figure 4-21.

Notre module de suivi de stocks est maintenant pratiquement terminé… à un détail près : le rechargement de la liste des produits en cas de changement de famille. Nous allons traiter ce point dans la section suivante.

**Figure 4–21** La page « historique du stock »

# Rendre la page interactive grâce à la gestion des événements

Une des grandes nouveautés apportées par ASP.NET est la gestion des événements : ce mécanisme, qui permet d'associer facilement un gestionnaire à un événement donné, améliore l'organisation du code et rend très simple la création de pages Web interactives.

Il existe deux types d'événements :

- les événements liés à la page ;
- les événements déclenchés par les contrôles.

Après une description du mécanisme des événements liés à la page (dont nous avons déjà vu des exemples : en particulier, le fameux `Page_Load`), nous allons voir comment il est possible d'implémenter le rechargement de la page `stocks.aspx` lorsque la sélection de la famille de produits est modifiée.

## Mieux structurer le code au sein d'une page grâce aux événements prédéfinis

Au sein d'une page ASP.NET, le code est organisé en gestionnaires d'événements, autrement dit en fonctions qui s'exécutent lorsqu'un événement donné survient.

Lors du chargement d'une page, un certain nombre d'événements se produisent :

| Événement | Description |
|---|---|
| `Page_Init` | Déclenché après l'initialisation de la page (à ce stade, les contrôles ne sont pas encore chargés) |
| `Page_Load` | Déclenché après le chargement de la page (à ce stade, les valeurs des contrôles ont été mises à jour à partir des valeurs contenues dans `__VIEWSTATE`) |
| `Page_Unload` | Déclenché lors de la désallocation de la page |

**__VIEWSTATE**

__VIEWSTATE est un contrôle HTML caché géré par ASP.NET, qui permet aux contrôles contenus dans la page de « garder leurs valeurs » lors d'un aller-retour de la page : nous décrivons son fonctionnement dans les pages qui suivent.

Pour associer un gestionnaire à un événement, le développeur doit placer dans le code une fonction de la forme suivante :

```
void TheObject_TheEvent (Object sender, EventArgs e)
```

où :

- `TheObject` désigne le nom de l'objet ayant déclenché l'événement ;
- `TheEvent` désigne le nom de l'événement ;
- `sender` est une variable de type `Object` qui pointe vers l'objet ayant déclenché l'événement ;
- `e` est une variable de type `EventArgs` qui contient des informations sur l'événement.

Il n'est pas nécessaire d'associer un gestionnaire à tous les événements de la page, mais uniquement à ceux qui sont utiles au développement (dans notre cas, `Page_Load` a pour l'instant suffi). Inversement, il est indispensable que tout code soit placé au sein d'un gestionnaire d'événement (ou au sein d'une fonction appelée directement ou indirectement depuis un gestionnaire).

Mais il y a mieux : des événements déclenchés par l'utilisateur (clic sur un bouton, changement de sélection dans une liste) peuvent provoquer un nouveau chargement de la page, avec au passage l'appel du gestionnaire d'événement associé. C'est le mécanisme des événements côté serveur que nous allons détailler maintenant.

**Comparaison avec ASP**

Dans le cas d'une page ASP, l'intégralité du code était exécutée lors du chargement de la page et il était nécessaire d'employer des `<%If…Then%>` pour exécuter spécifiquement un bloc donné en fonction des circonstances. De plus, un événement ne pouvait pas provoquer le rechargement de la page, comme c'est désormais le cas avec ASP.NET

## Gérer le changement de famille grâce aux événéments déclenchés par DropDownList

Nous souhaitons ajouter de l'interactivité à la page qui affiche l'état des stocks classés par famille de produits : si l'utilisateur modifie la sélection de la famille, nous souhaitons que la liste des produits soit rechargée en conséquence.

Pour cela, pas de miracle : il faut détecter le changement de sélection dans la liste et, lorsqu'il survient, faire une nouvelle requête vers le serveur en demandant le rechargement de la page, actualisée en fonction de la nouvelle famille de produits sélectionnée.

Pour accomplir ceci avec une technologie classique comme ASP, il aurait fallu implémenter, côté client, un gestionnaire en langage de script (JavaScript) associé à l'événement `onChange` de la liste des familles, qui provoque le rechargement de la page.

Dans cet exemple, on fait l'hypothèse que la page associée à l'attribut `action` du formulaire est la page qui contient le formulaire (le formulaire « s'auto-poste »).

```
<select onChange="document.forms[0].submit()"></select>
```

Puis, il aurait fallu implémenter le remplissage de la liste des familles, la sélection de la nouvelle famille (obtenue à partir des données postées) et le chargement de l'état des stocks correspondants.

Désormais, tout ce travail est effectué automatiquement par ASP.NET grâce aux gestionnaires d'événements serveur.

Pour associer un gestionnaire au changement de sélection dans la liste des familles, il suffit de modifier comme suit le contenu HTML de la page `stocks.aspx` :

**Stocks.aspx (contenu HTML)**

```
<asp:DropDownList id="ListeFamilles" runat="server"
   AutoPostBack="true"
   OnSelectedIndexChanged="OnFamilleChanged"/>
```

Détaillons les modifications effectuées :

- L'instruction `OnSelectedIndexChanged="OnFamilleChanged"` associe le gestionnaire `OnFamilleChanged` (nom choisi par le développeur) à l'événement « changement de sélection ».
- L'instruction `AutoPostBack="True"` indique au moteur ASP.NET qu'il doit générer automatiquement le script client qui provoquera le rechargement de la page lors d'un changement de famille (notons que ce script sera adapté aux possibilités du navigateur en fonction de sa version).

Dans la page `stocks.aspx`, implémentez le gestionnaire d'événement `OnFamille Changed`, qui effectue la mise à jour du tableau `EtatStock` (on présente seulement la version VB.NET) .

**Stocks.aspx – Fonction OnFamilleChanged (version VB.NET)**

```
Sub OnFamilleChanged(sender As Object, e As EventArgs)
   AfficherStocksProduits()
End Sub
```

Enfin, modifiez la fonction `Page_Load`, de manière à ne pas effectuer d'opération inutile lors du rechargement de la page (la propriété `IsPostBack` indique que la page a été postée vers le serveur, suite à un événement) :

**Stocks.aspx – Fonction Page_Load (version VB.NET)**

```
Sub Page_Load(sender As Object, e As EventArgs)
   If Not IsPostBack Then
        ChargerListeFamilles()
        AfficherStocksProduits()
   End If
End Sub
```

**Stocks.aspx – Fonction Page_Load (version C#)**

```
void Page_Load(Object sender, EventArgs e)
{
  if(IsPostBack==false)
  {
      ChargerListeFamilles();
      AfficherStocksProduits();
   }
}
```

Testez la page… un changement de sélection dans la liste des familles de produits doit désormais provoquer la mise à jour du tableau d'état des stocks !

## Comprendre le mécanisme de gestion événementielle d'ASP.NET

Pour mieux comprendre le mécanisme de gestion événementielle, détaillons ce qui se passe lors de cet aller-retour (*round trip*) vers le serveur, illustré à la figure 4-22.

Si vous regardez le fichier source de la page affichée dans votre navigateur, vous y trouverez un script provoquant l'envoi du formulaire au serveur lors de la sélection de la famille de produits :

```
<select id="ListeFamilles"
 onchange="__doPostBack('ListeFamilles','')" >
```

La fonction __doPostBack réalise l'envoi du formulaire (qui conduit au rechargement de la même page ; voir l'attribut action de la balise <form>)

```
function __doPostBack(eventTarget, eventArgument) {
   var theform = document._ctl0;
   theform.__EVENTTARGET.value = eventTarget;
   theform.__EVENTARGUMENT.value = eventArgument;
   theform.submit();
}
```

Grâce à deux contrôles (__EVENTTARGET et __EVENTARGUMENT), la source de l'événement et les éventuels arguments sont transmis au serveur, qui peut ainsi noter qu'il s'agit d'une page postée (mise à jour d'IsPostBack), déterminer le gestionnaire d'événement à appeler et lui passer les arguments nécessaires.

**Sans AutoPostBack**

Si on ne spécifie pas AutoPostBack="True", aucun script client ne sera généré : un changement de sélection ne provoquera donc pas le rechargement « automatique » de la page ; néanmoins, il provoquera l'appel du gestionnaire OnFamilleChanged lors du prochain rechargement de la page, grâce à la comparaison entre les valeurs des champs de formulaire et celles stockées dans ViewState.

La page contient un troisième contrôle caché nommé __VIEWSTATE qui contient une copie (codée sous une forme synthétique) des valeurs des contrôles serveur (dans notre cas : la liste des familles et le contenu du tableau d'état du stock) :

```
<input type="hidden" name="__VIEWSTATE"
 value="dDwtNTc5NDY2NTIyO3Q8O2w8aT… " />
```

Le rôle de __VIEWSTATE est de permettre l'initialisation des contrôles dans la page nouvellement chargée à partir des valeurs existantes des contrôles : arrivée sur le serveur, la requête HTTP provoque une nouvelle exécution du programme « cracheur de HTML » associé à la page (déjà compilé et maintenu en cache), lequel décode les valeurs contenues dans __VIEWSTATE et initialise en conséquence les contrôles serveur. En d'autres termes, on a l'impression de charger la même page, alors qu'en réalité, on charge une nouvelle instance de la page initialisée avec les mêmes valeurs !

Après cette phase d'initialisation, le gestionnaire Page_Load est appelé. Grâce à la propriété IsPostBack, il est possible de contrôler spécifiquement les données à recharger : en l'occurrence, le contenu de la liste des familles ayant été initialisé à partir des valeurs contenues dans __VIEWSTATE, il n'est plus nécessaire de se connecter à la base pour récupérer la liste des familles.

Ensuite, l'examen de la valeur du contrôle caché __EVENTTARGET conduit à l'appel du gestionnaire OnFamilleChanged, qui réalise, dans notre cas, le chargement de l'état de stocks associé à la nouvelle famille sélectionnée.

Remarquons au passage qu'ici, il n'est pas nécessaire de transporter la valeur du DataGrid dans __VIEWSTATE, puisqu'il sera systématiquement rechargé à partir de la base :

```
<asp:DataGrid id="EtatStock" runat="server" …
 EnableViewState="false">
```

Enfin, la page est renvoyée vers le navigateur !

**À RETENIR EnableViewState**

La propriété EnableViewState permet d'exclure spécifiquement un contrôle serveur du mécanisme de ViewState. Ceci permet d'optimiser la taille des pages HTML générées par le moteur ASP.NET.

En résumé, ce mécanisme de gestion des événements facilite l'implémentation de pages Web interactives et conduit à une meilleure structuration du code, tout en restant relativement peu coûteux en termes de performances, à condition de tirer parti des possibilités d'optimisation offertes par IsPostBack et EnableViewState.

Enfin, notons que cette architecture ne stocke aucune donnée associée au client sur le serveur (l'information est stockée dans les contrôles cachés de la page HTML) : par conséquent, plusieurs requêtes successives d'un même client peuvent être traitées par des serveurs différents, ce qui est appréciable dans le cas de grappes de serveurs.

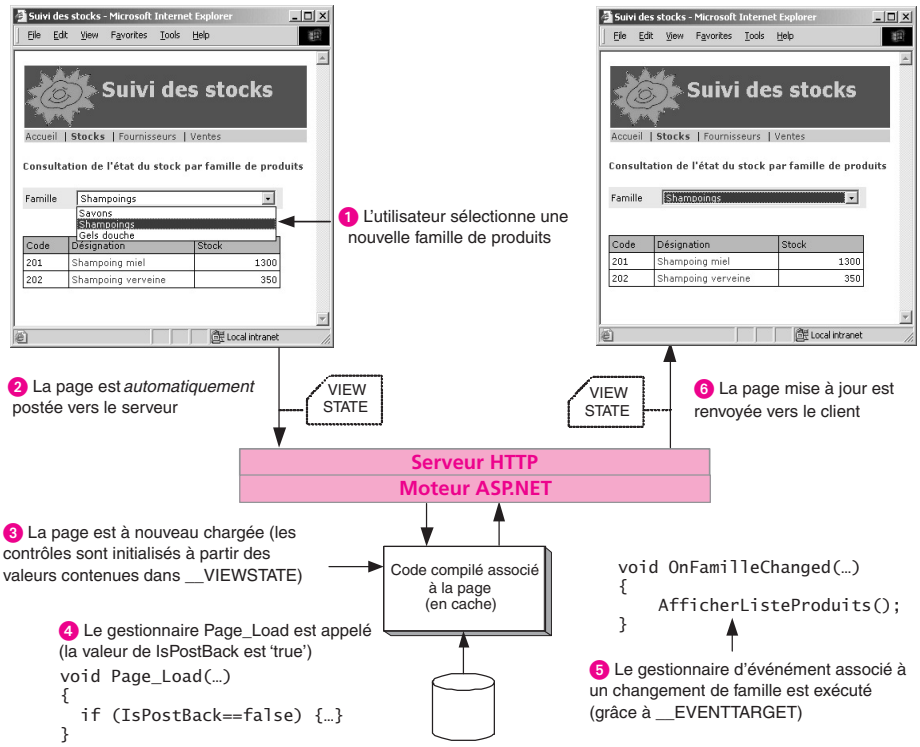



**Figure 4–22** Le mécanisme de gestion événementielle

# En résumé…

Dans ce chapitre, nous avons présenté plusieurs éléments fondamentaux d'ASP.NET, que nous avons mis en œuvre pour réaliser le module de suivi des stocks de notre étude de cas :

- la bibliothèque ADO.NET, qui permet de gérer la communication avec des sources de données et qui a été conçue pour obtenir un code mieux organisé (séparation nette entre les classes de manipulation et les classes de communication) et mieux adapté aux architectures Web (conservation de données en cache du fait du caractère déconnecté des applications, gestion de sources de données XML) ;
- la technique permettant de partager la connexion à la base de données entre tous les éléments de l'application, via le fichier `global.asax` et l'objet `Session` ;
- le mécanisme de liaison d'un contrôle serveur à une source de données via la propriété `DataSource` avec l'aide des classes `SqlConnection`, `SqlCommand` et `SqlDataReader` (variante possible avec `SqlDataAdapter` et `DataTable`) ;

- les nombreuses possibilités de paramétrage du contrôle `DataGrid` (colonnes liées à un champ, colonnes hypertextes, contrôle des couleurs et de la largeur des colonnes, contrôle des bordures du tableau) ;
- le mécanisme de gestion événementielle d'ASP.NET, fondé sur la technique de l'aller-retour (*round trip*), détecté grâce à la propriété `IsPostBack` et la conservation des valeurs des contrôles assurée par `__VIEWSTATE`.

Dans le chapitre suivant, nous allons implémenter le module de gestion des fournisseurs de notre étude de cas : nous verrons au passage comment mettre à jour une base de données et valider les informations saisies par un utilisateur.

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

# 5

ASP.NET

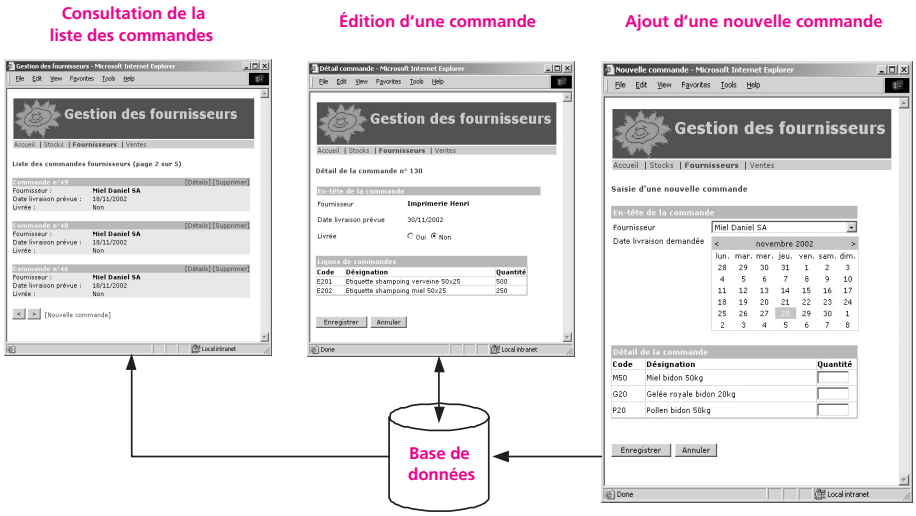
Repeater | Event bubbling | PagedDataSource | SqlDataAdapter | SqlCommandBuilder | TemplateColumn

## SOMMAIRE

- Affichage de la liste des commandes avec un contrôle Repeater
- Ajout d'un mécanisme de pagination à la liste
- Création, modification et suppression de commandes
- Validation des informations saisies par l'utilisateur
- Gestion de transactions

## 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.

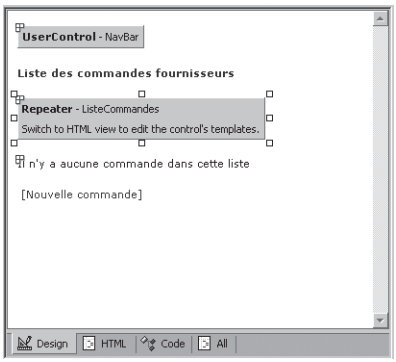



**Figure 5–2** Maquette de la page de consultation des commandes

```
        <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`).

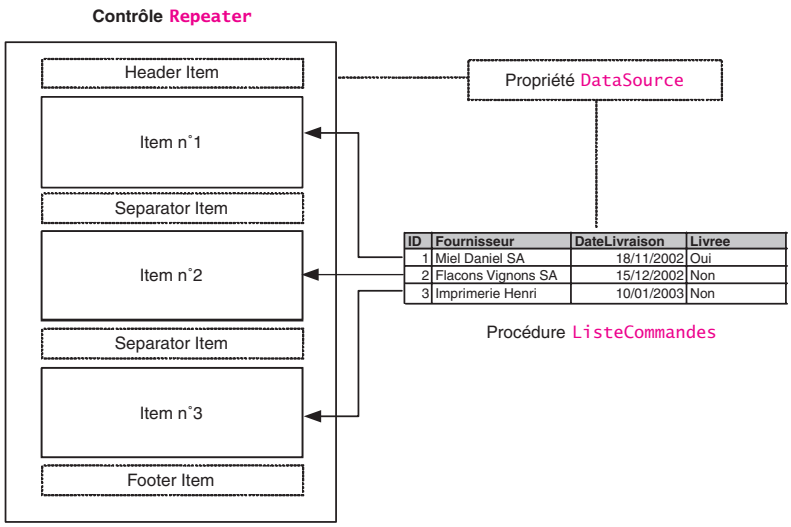



**Figure 5–3** Architecture du contrôle Repeater

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.

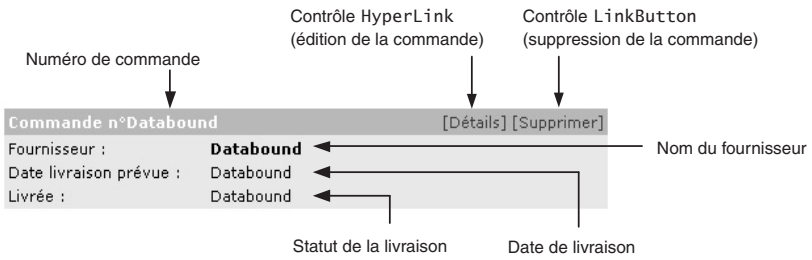



**Figure 5–4** Maquette de l'élément `Item` du `Repeater`

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).

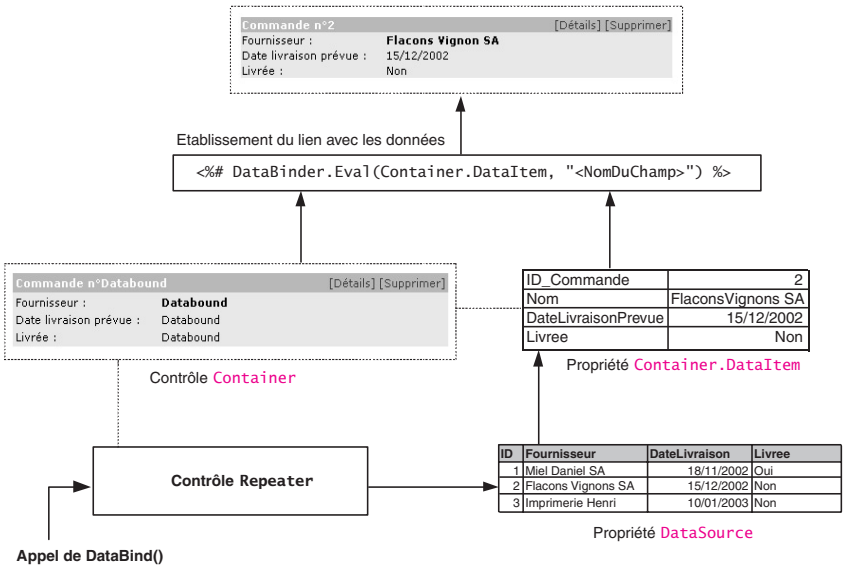



**Figure 5–5** Mécanisme de lien aux données pour le contrôle Repeater

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.

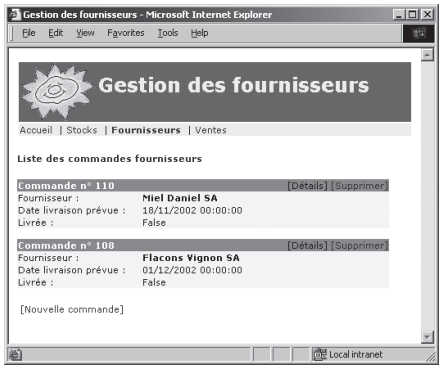



**Figure 5–6**
Affichage de la liste des commandes

## 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
```

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 précisement 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.

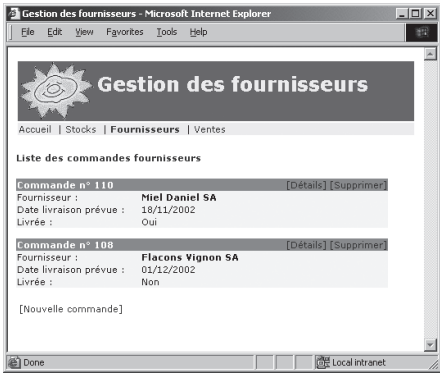



**Figure 5–7**
La liste des commandes après formatage

## 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.

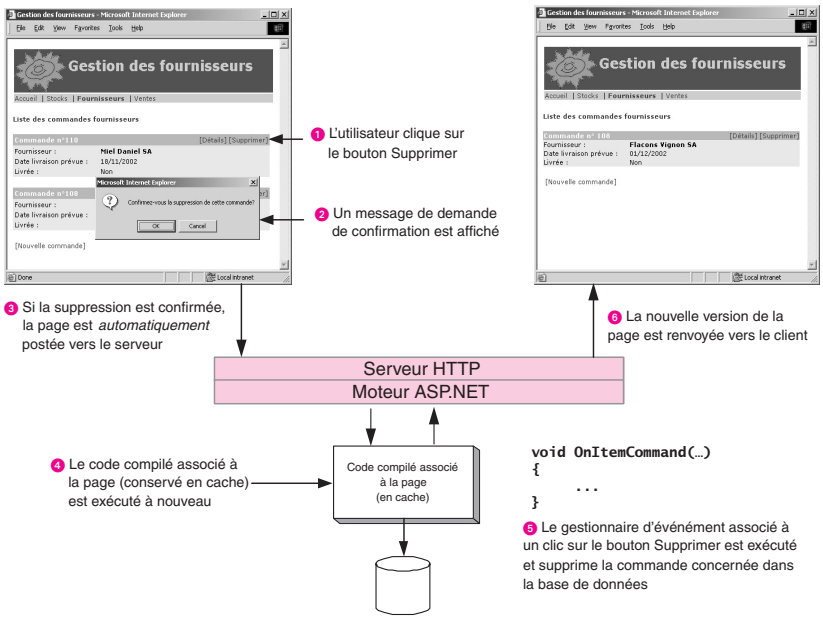



**Figure 5–8** Cinématique de la suppression d'une commande

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).

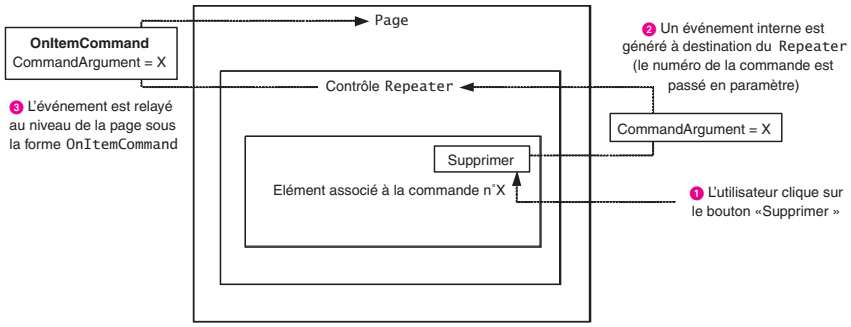



**Figure 5–9** Mécanisme de remontée des événements depuis un contrôle interne

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
```

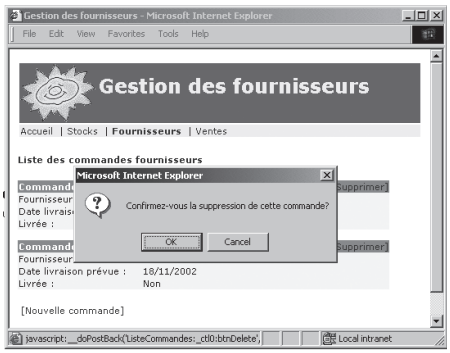



**Figure 5–10** Confirmation de la suppression d'une commande

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

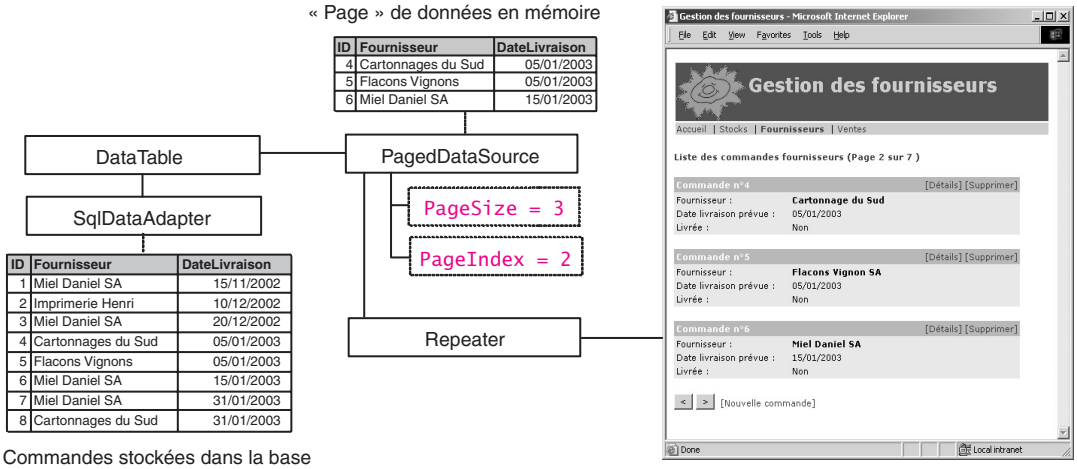



**Figure 5–11** Mise en place d'une source de données paginée

## 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).

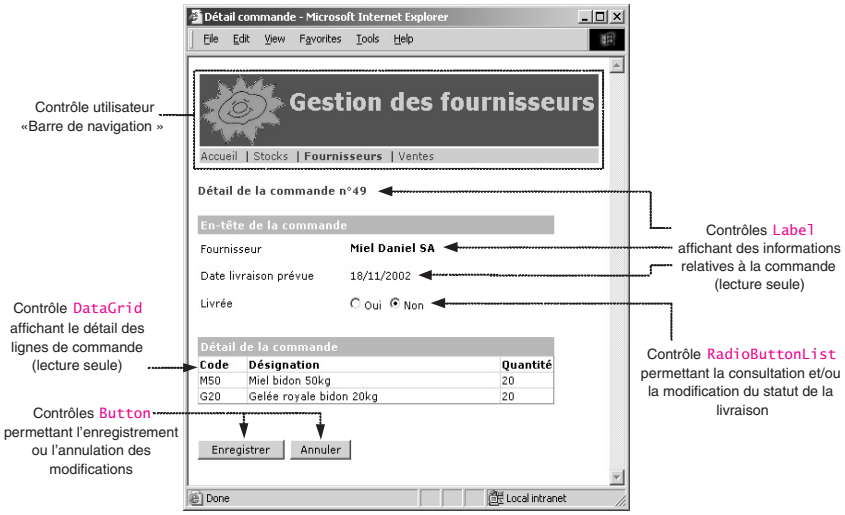



**Figure 5–12** La page de consultation du détail d'une commande

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.

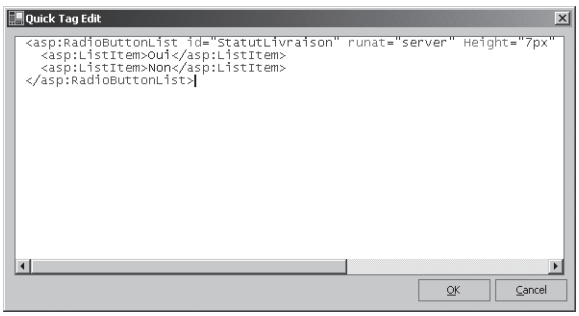



**Figure 5–13** La fonctionnalité Quick Tag Edit de Web Matrix

**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` |

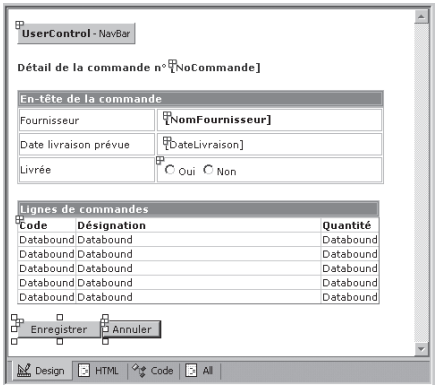



**Figure 5–14** Maquette de la page Détails de la commande

### 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 `OnEnregistrerClicked` 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)
{
...
}
```

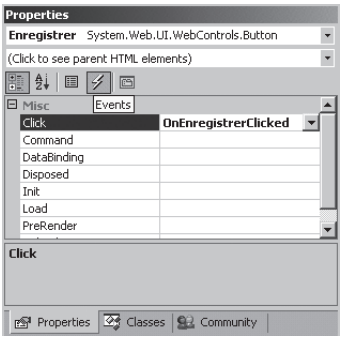



**Figure 5–15** La feuille de propriétés Événements

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`).

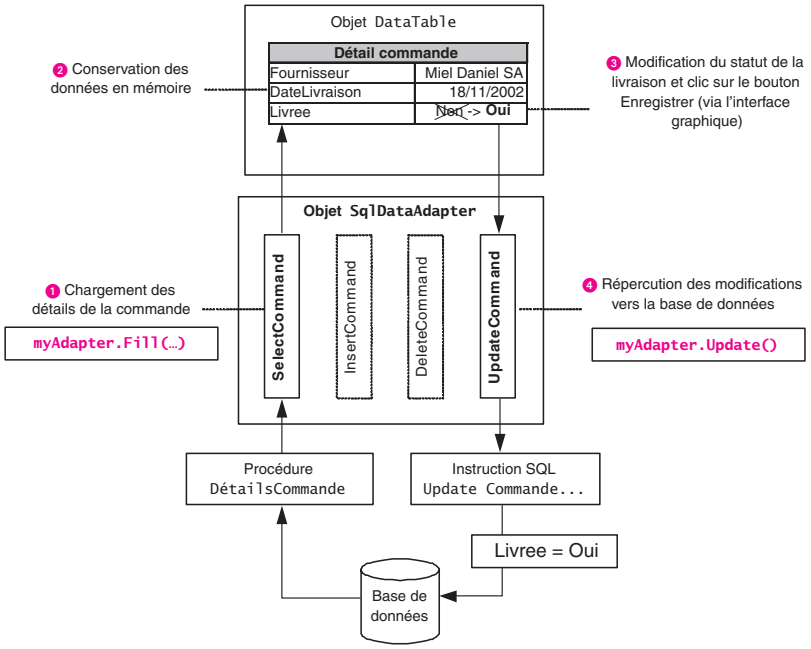



**Figure 5–16** Consultation et mise à jour d'une commande avec SqlDataAdapter

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.

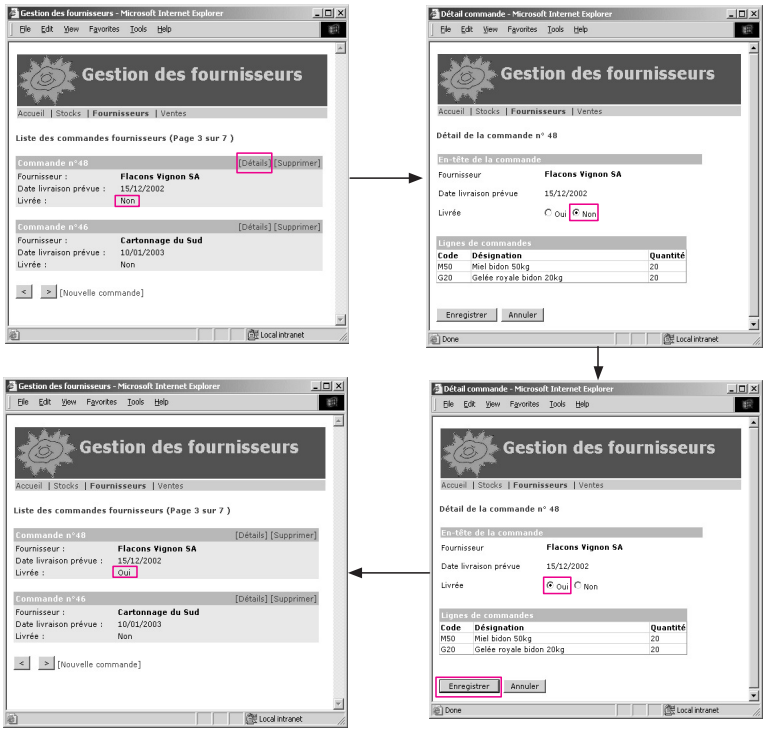



**Figure 5–17** Processus de consultation et modification d'une commande

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).

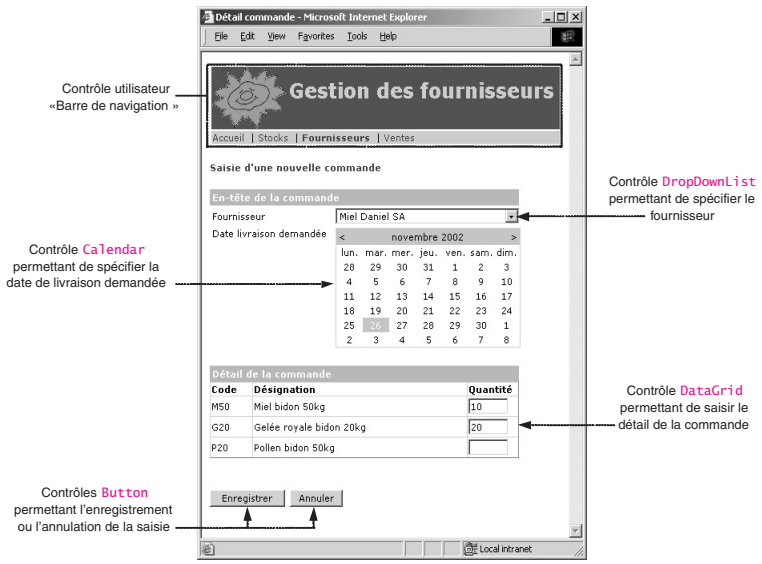



**Figure 5–18** La page de saisie d'une nouvelle commande

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).

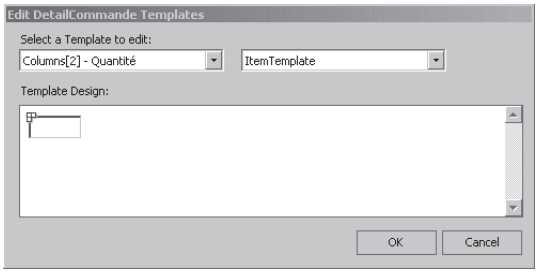



**Figure 5–19** La boîte de dialogue Edit Template

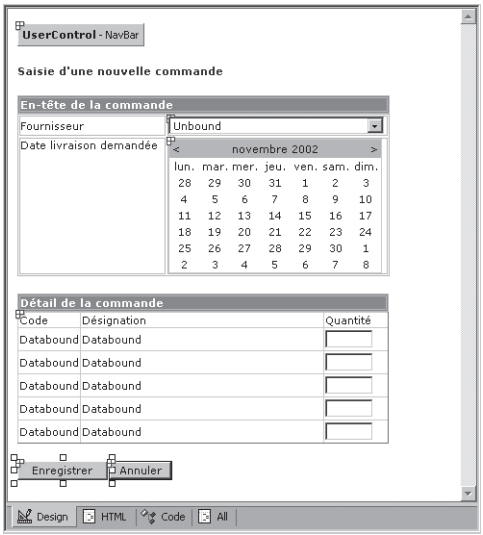



**Figure 5–20** Maquette de la page de saisie d'une nouvelle commande

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`).

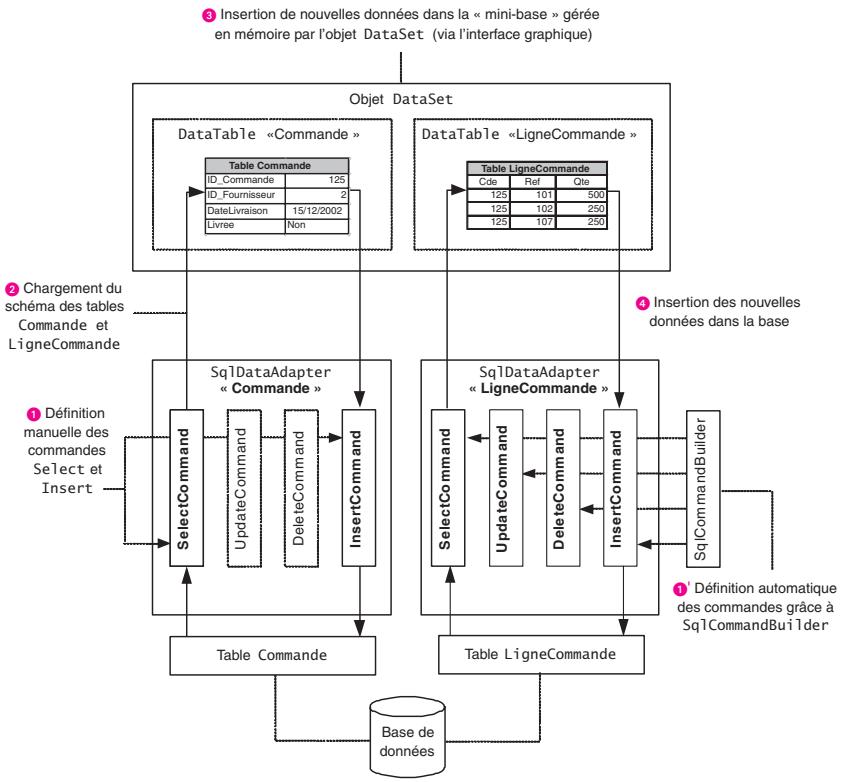



**Figure 5–21** Création d'une nouvelle commande avec SqlDataAdapter

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

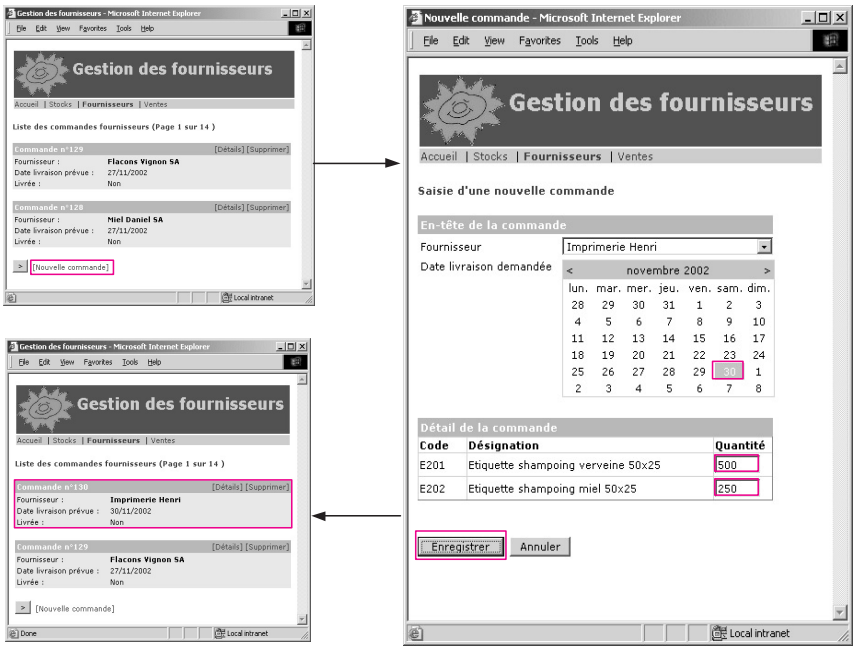



**Figure 5–22** Mécanisme de création d'une nouvelle commande

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).

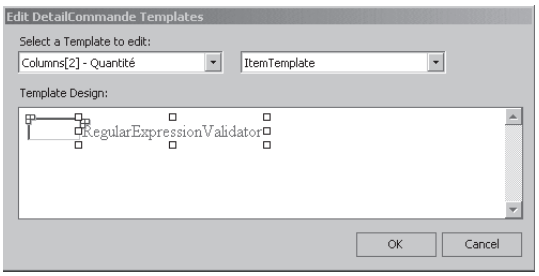



**Figure 5–23** Ajout d'un contrôle de validation

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`).

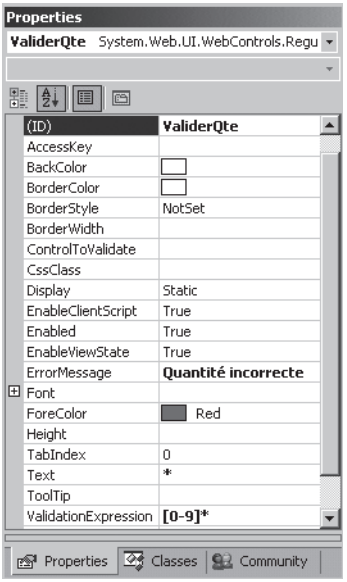



**Figure 5–24** Propriétés du contrôle de validation

### 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.

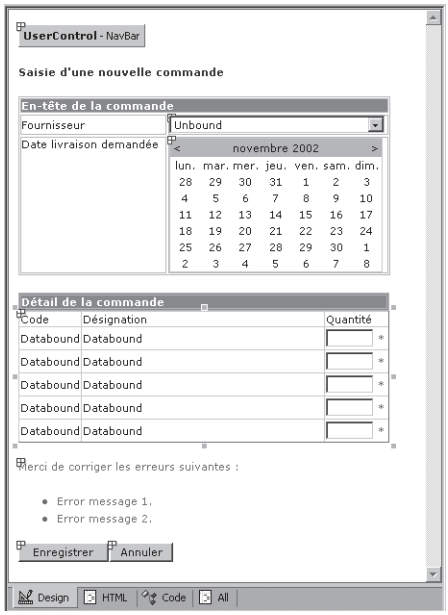



**Figure 5–25** Ajout d'un contrôle de type `ValidationSummary`

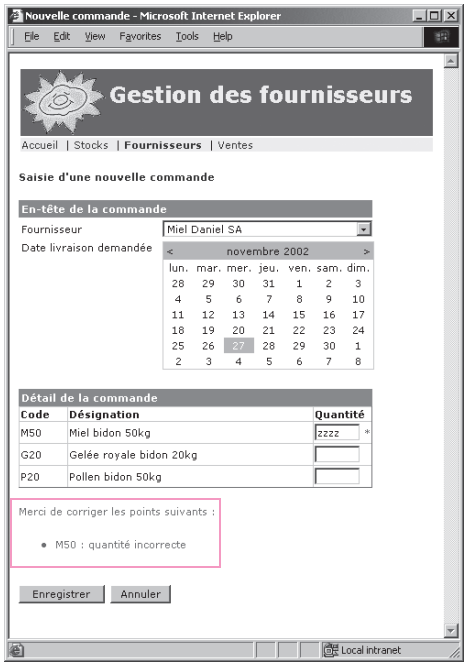



**Figure 5–26** Détection d'une erreur de saisie

### 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

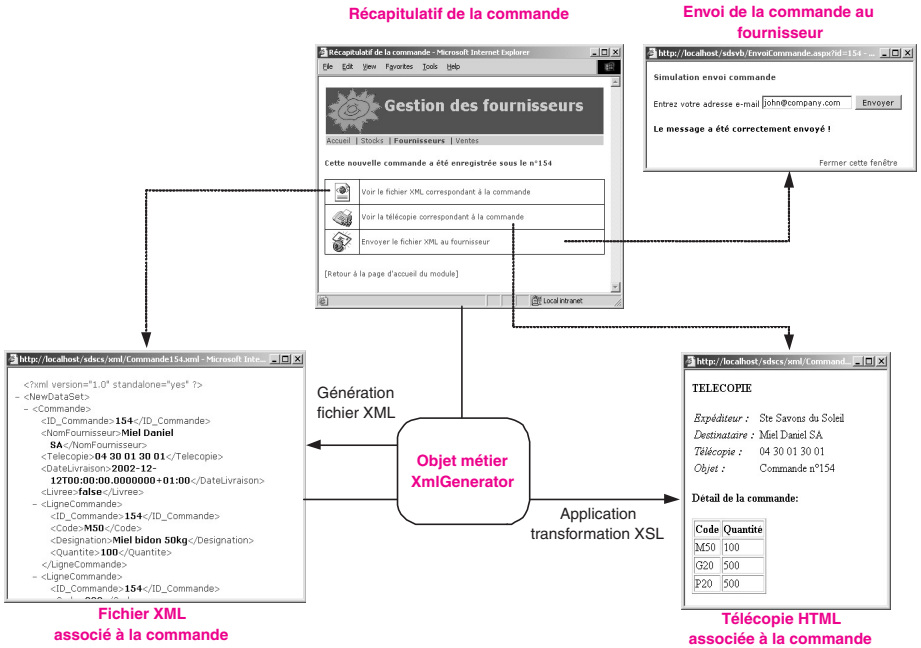
Compilateurs vbc et csc | WriteXml | XmlDocument | XslTransform | MailMessage | SmtpMail

## SOMMAIRE

- Création d'un objet métier
- Génération de documents XML
- Application de transformations XSL
- Envoi de fichiers par messagerie

## 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.

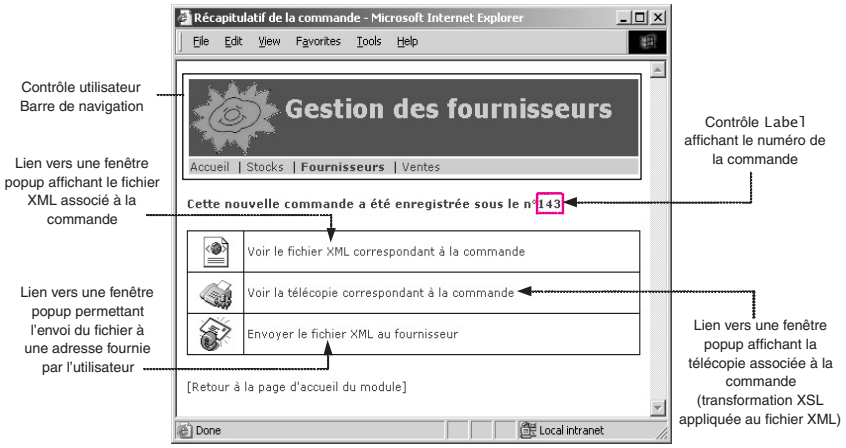



**Figure 6–1** La page récapitulative de la commande

**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.

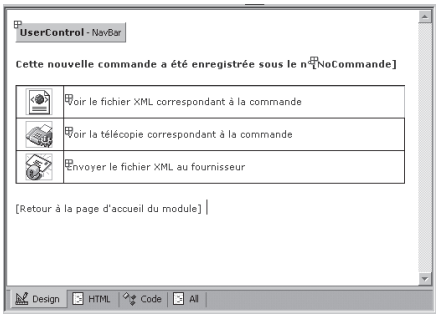



**Figure 6–2** Maquette de la page récapitulative d'une commande

**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).

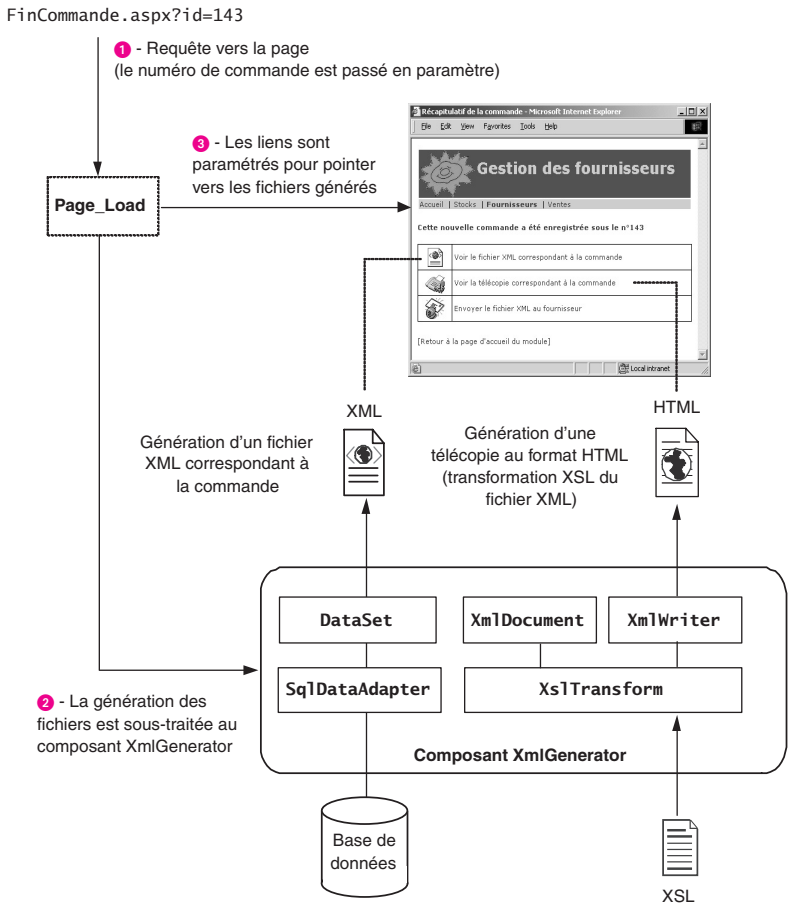



**Figure 6–3** Architecture de la page récapitulative d'une commande

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).

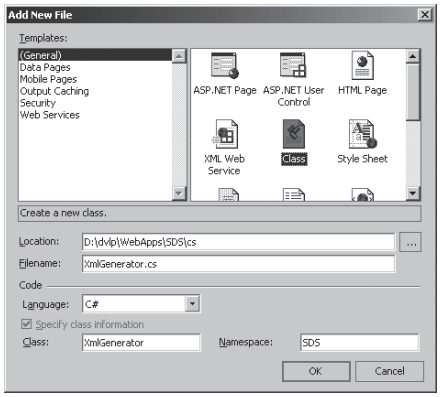



**Figure 6–4** Création d'une classe avec Web Matrix

**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.

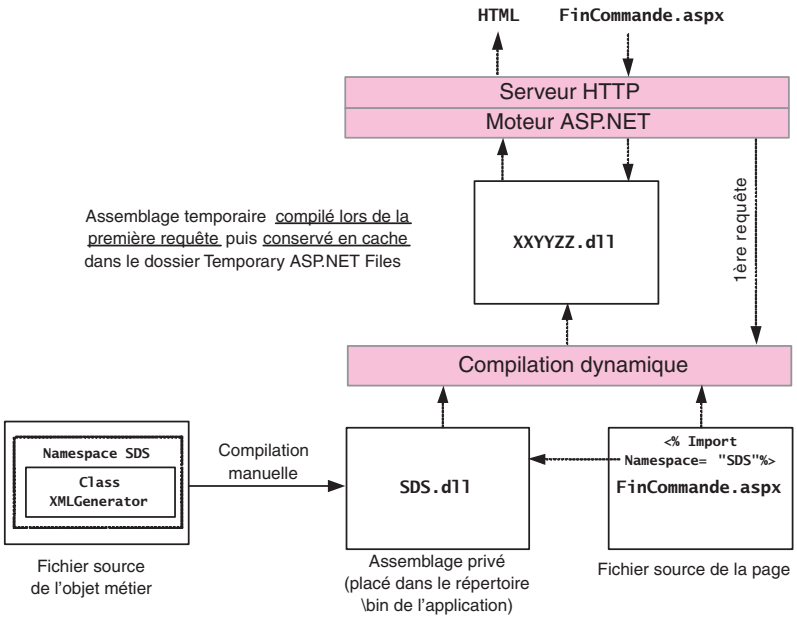



**Figure 6–5** Processus de compilation d'une page utilisant un composant externe

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).

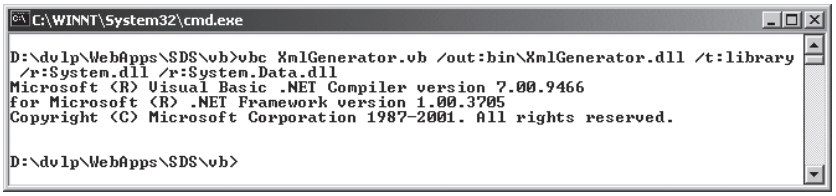



**Figure 6–6** Compilation d'une classe

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).

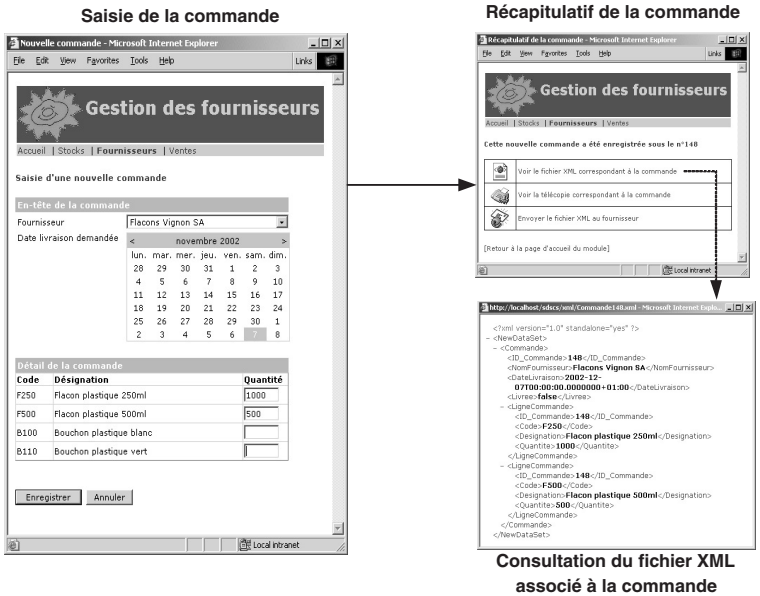



**Figure 6–7** Consultation du fichier XML associé à la commande

**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.

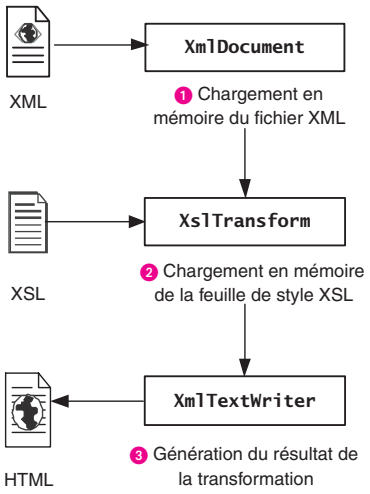



**Figure 6–8** Application d'une transformation XSL

**Fichier GenererTelecopie.xsl**

```
<?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>
```

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.

**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.

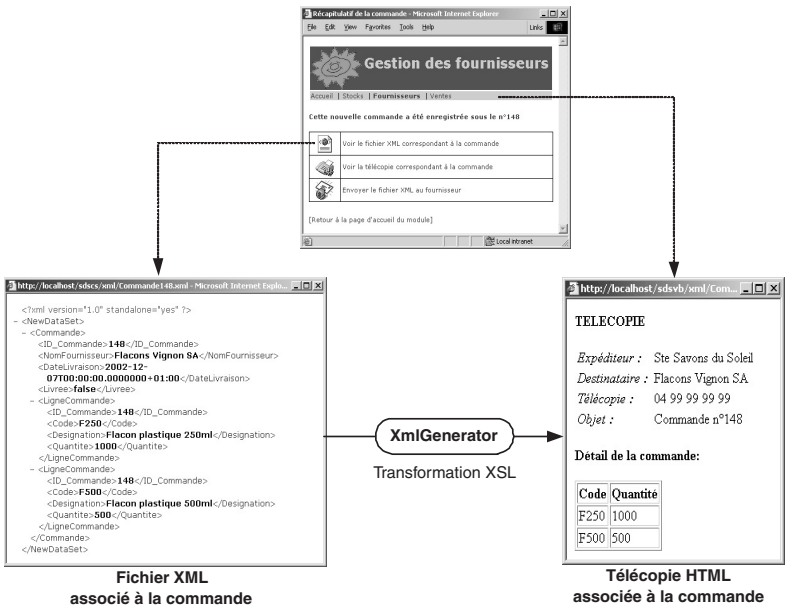



**Figure 6–9** Consultation de la télécopie générée via une transformation XSL

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).

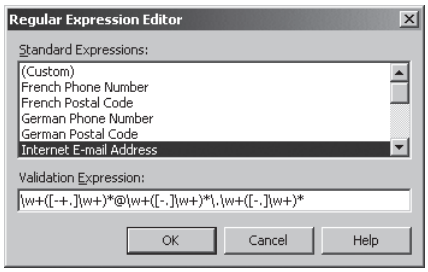



**Figure 6–10** Insertion d'une expression régulière

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

**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>
```

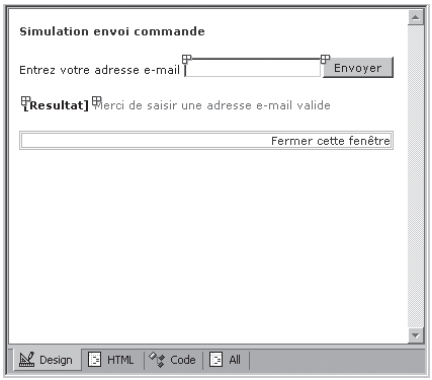



**Figure 6–11** Maquette de la page d'envoi d'une commande

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.

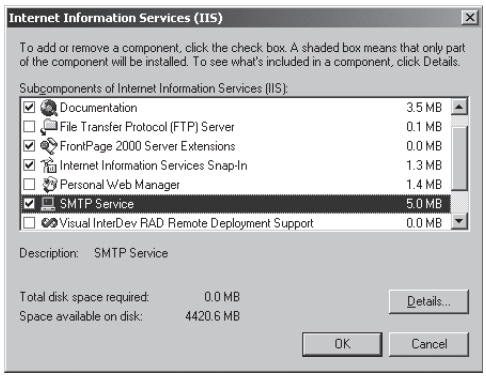



**Figure 6–12** Installation du serveur SMTP d'IIS

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.

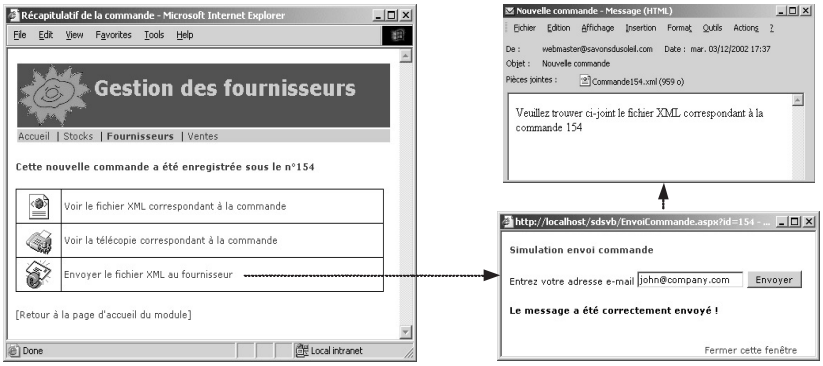



**Figure 6–13** Envoi d'un message contenant le fichier associé à la commande

# 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

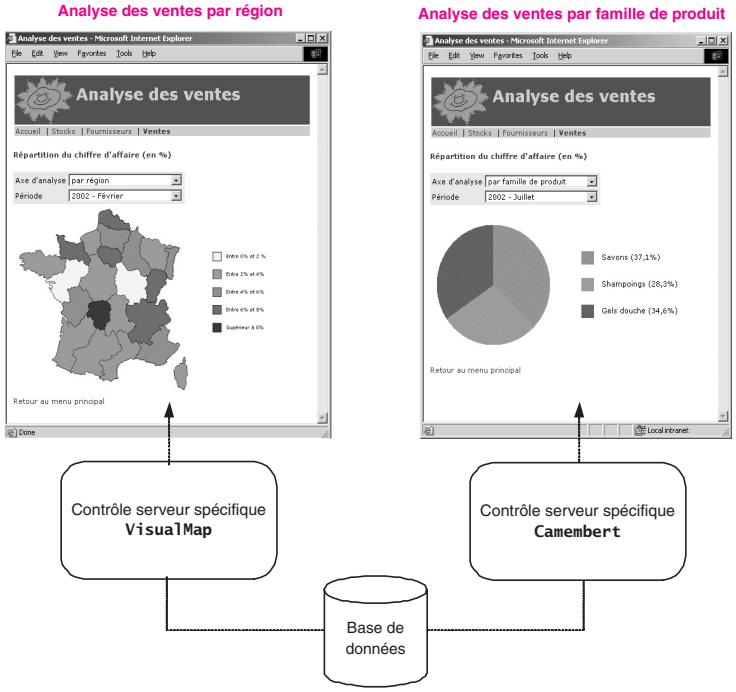
Contrôle serveur spécifique | Global Assembly Cache (GAC) | WebControl | Render | System.Drawing

## SOMMAIRE


- Utilisation de contrôles serveur spécifiques
- Consultation des ventes par région avec VisualMap
- Création d'un contrôle serveur spécifique « camembert »


## 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.

**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).

# 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 répartition 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.

## 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) :

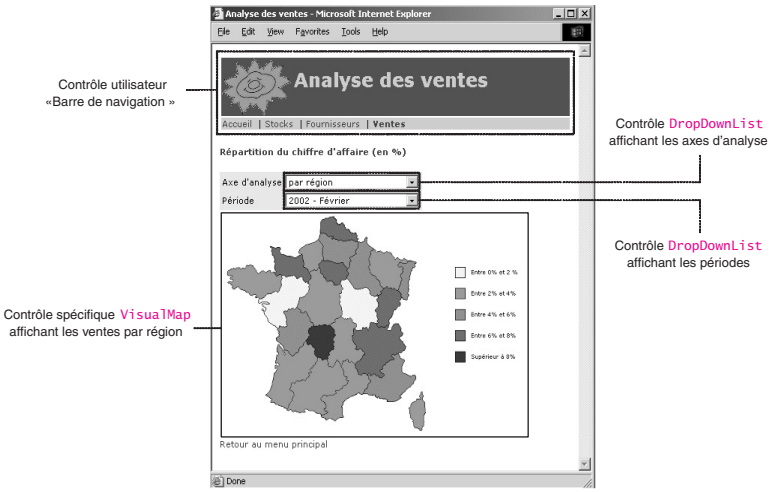



**Figure 7–1** La page d'analyse des ventes par région

- 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`

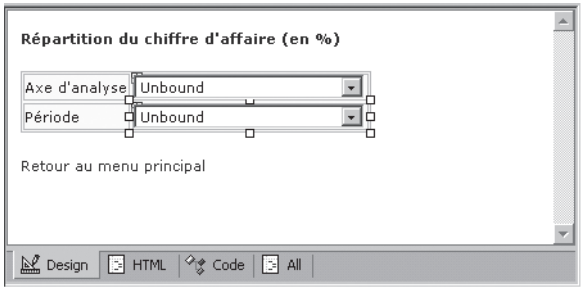



**Figure 7–2** Maquette de la page d'analyse des ventes

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).

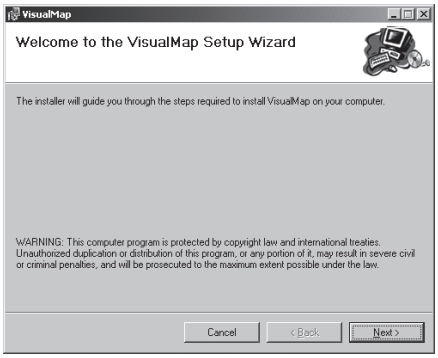



**Figure 7–3** Installation de VisualMap

### 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).

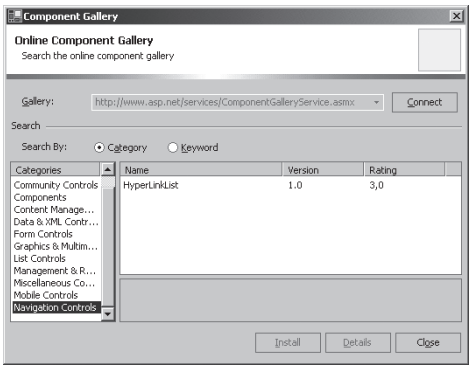



**Figure 7–4** Recherche de composants en ligne avec Web Matrix

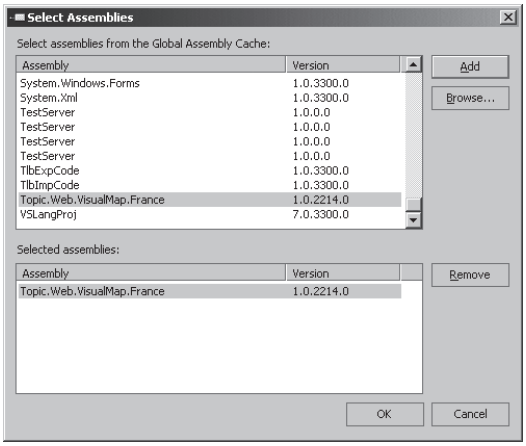



**Figure 7–5**
Ajout du composant VisualMap
à la barre d'outils de Web Matrix

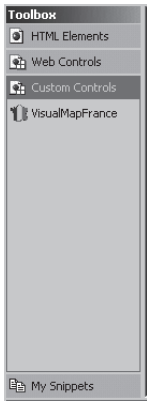



**Figure 7–6**
La barre d'outils Custom
Controls de Web Matrix

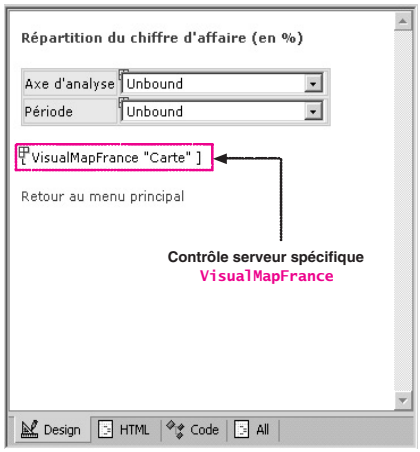



**Figure 7–7**
Maquette de la page d'analyse des ventes
après insertion du composant `VisualMap`

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).

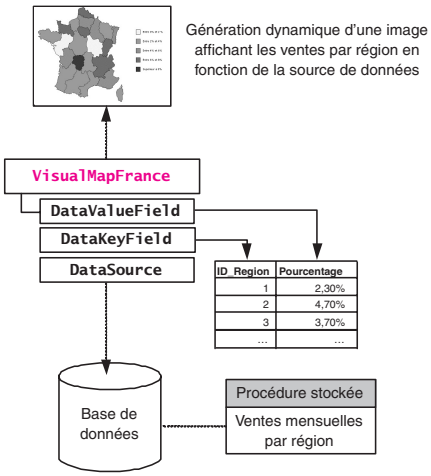



**Figure 7–8** Architecture du contrôle serveur `VisualMapFrance`

## 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

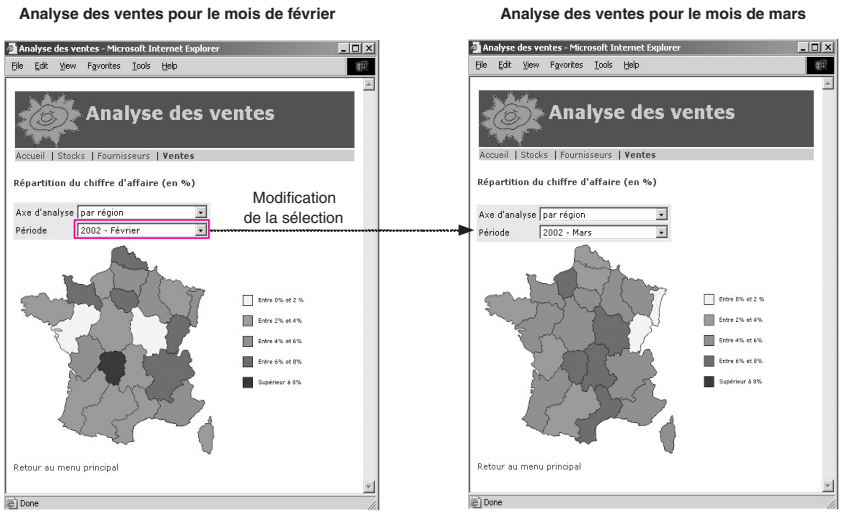



**Figure 7–9** Cinématique de la page d'analyse des ventes par régions

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.

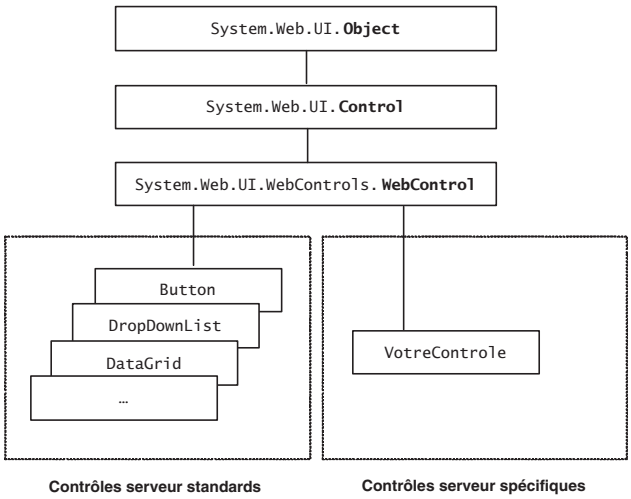



**Figure 7–10** Hiérarchies de classes utilisées pour les contrôles serveur

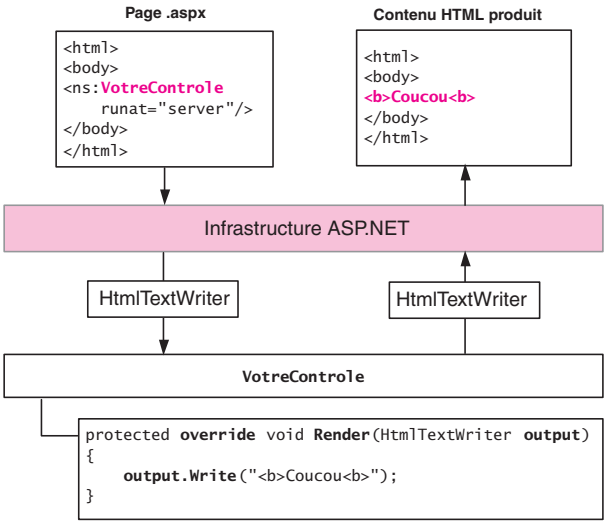



**Figure 7–11** Architecture interne d'un contrôle serveur

### 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).

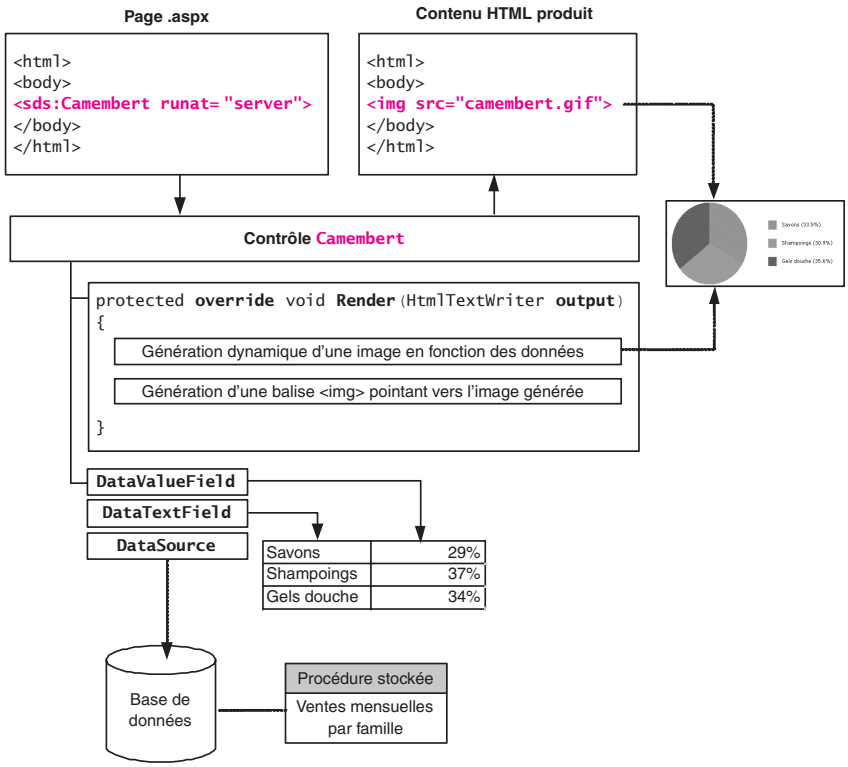



**Figure 7–12** Architecture du contrôle `Camembert`

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#)**

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

```
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);
}
```

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" %>
```

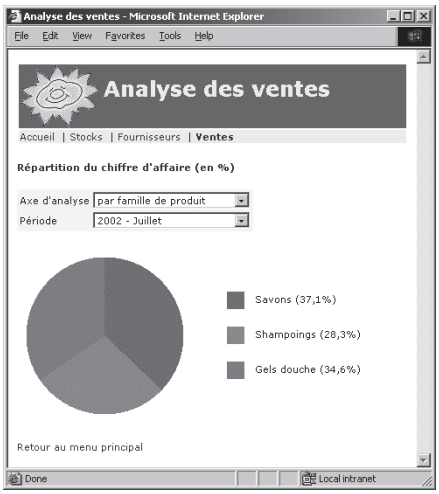



**Figure 7–13** Page d'analyse des ventes par produit

**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

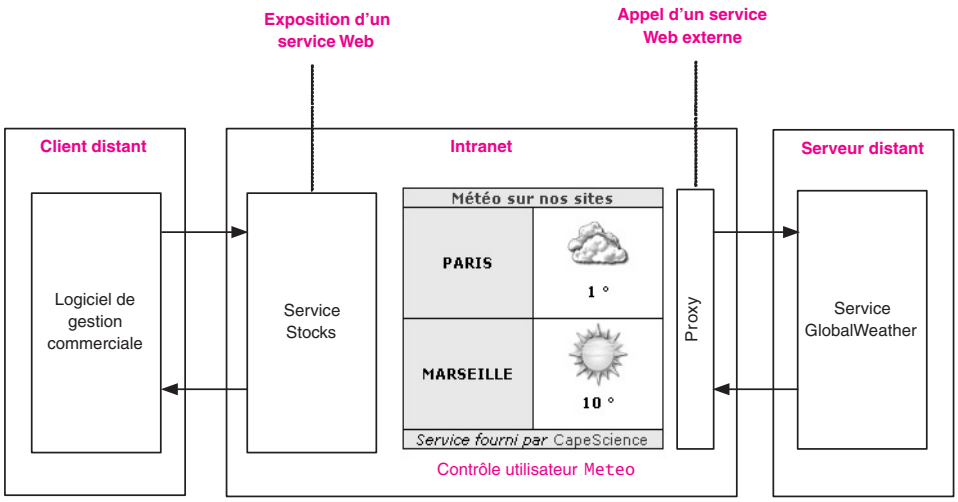
Service Web | Fichier .asmx | WSDL | UDDI | SOAP | Proxy

## 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.

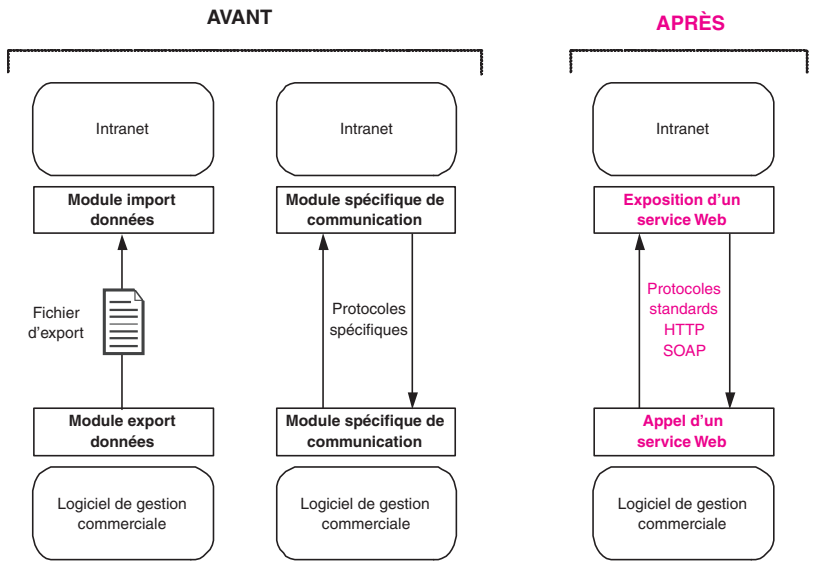



**Figure 8–1** Quelques scénarios d'échanges de données entre applications

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.

**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.

**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).

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.

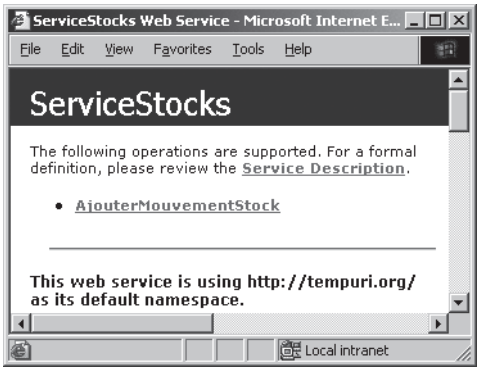



**Figure 8–2** Page d'accueil par défaut du service

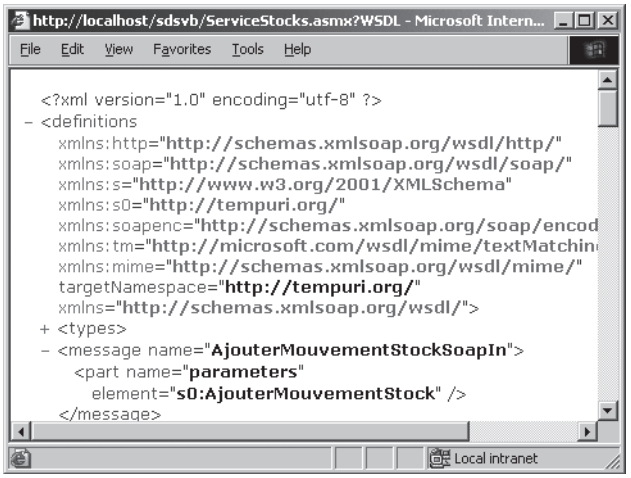



**Figure 8–3** Description du service au format WSDL

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).

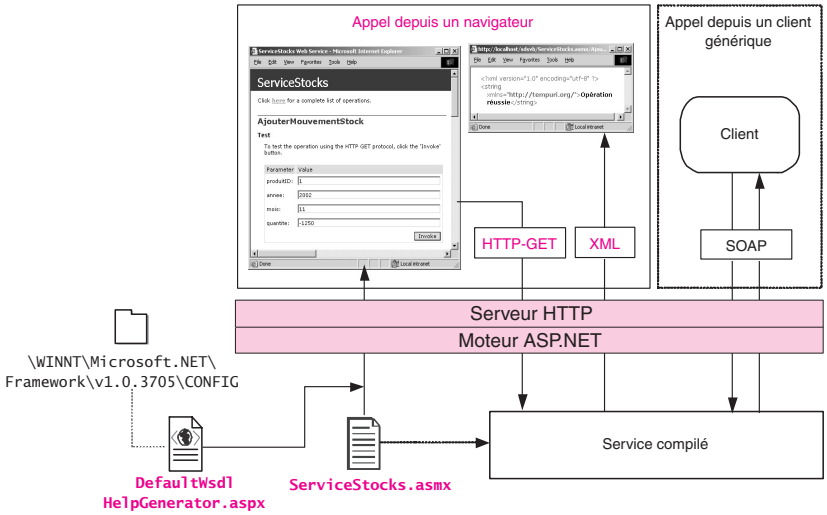



**Figure 8–4** Appel du service depuis différents types de clients

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.).

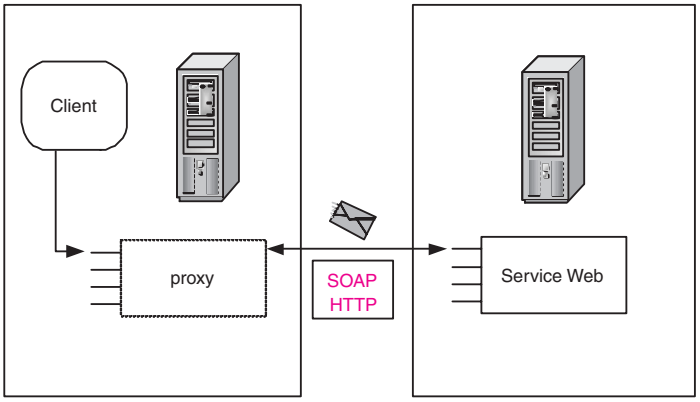



**Figure 8–5** Appel du service Web via un proxy

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)**

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 !).

```
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
```

```
    <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).

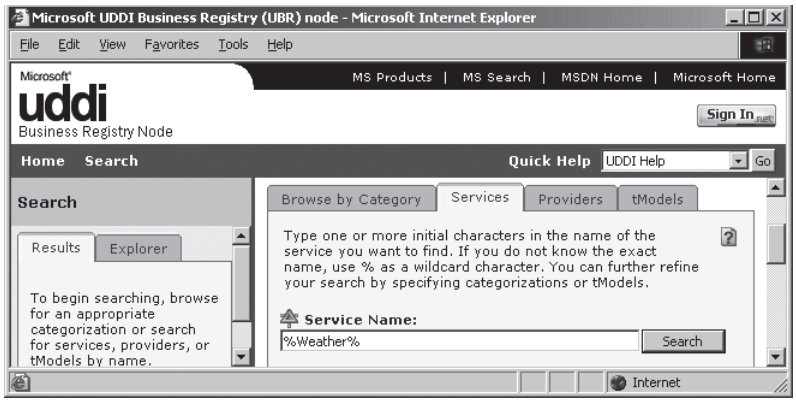



**Figure 8–6** Recherche d'un service Web à l'aide d'un annuaire UDDI

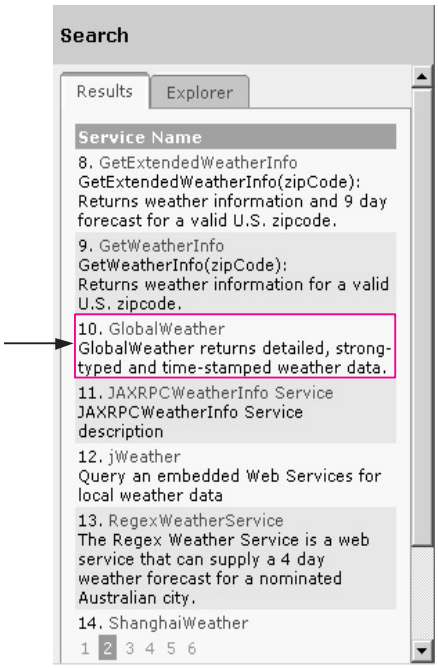



**Figure 8–7** Résultats de la recherche

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`).

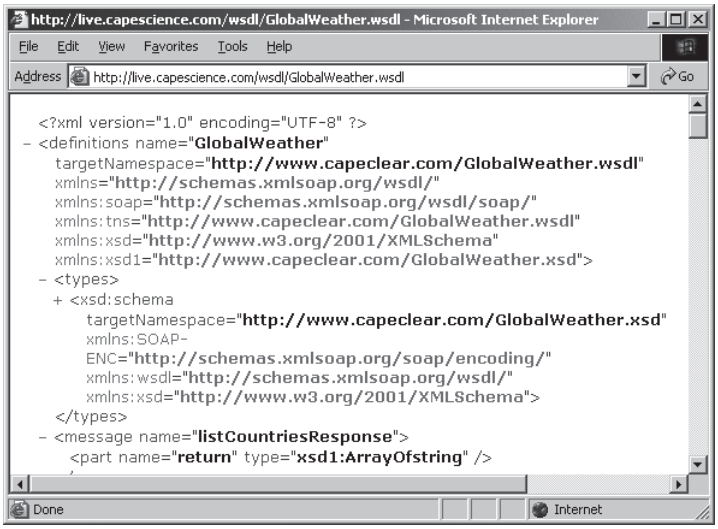



**Figure 8–8** Fichier WSDL associé au service GlobalWeather

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).

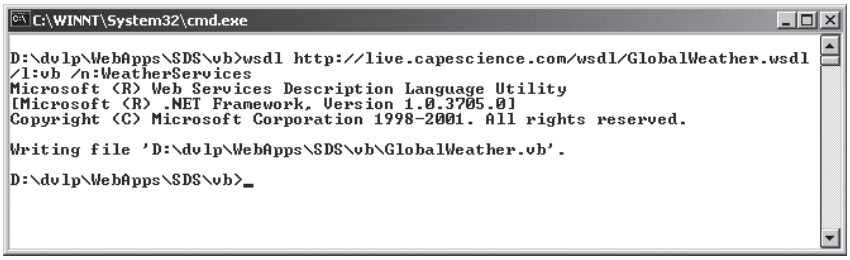



**Figure 8–9** Génération d'un proxy avec WSDL

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**

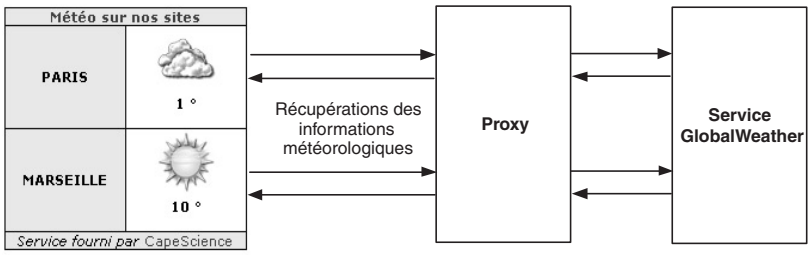
Un contrôle utilisateur est un morceau de page Web encapsulé dans un composant graphique réutilisable (voir chapitre 3).

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 !)

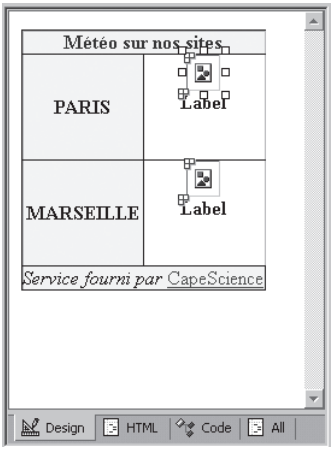



**Figure 8–11** Maquette du contrôle utilisateur Meteo

## 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.

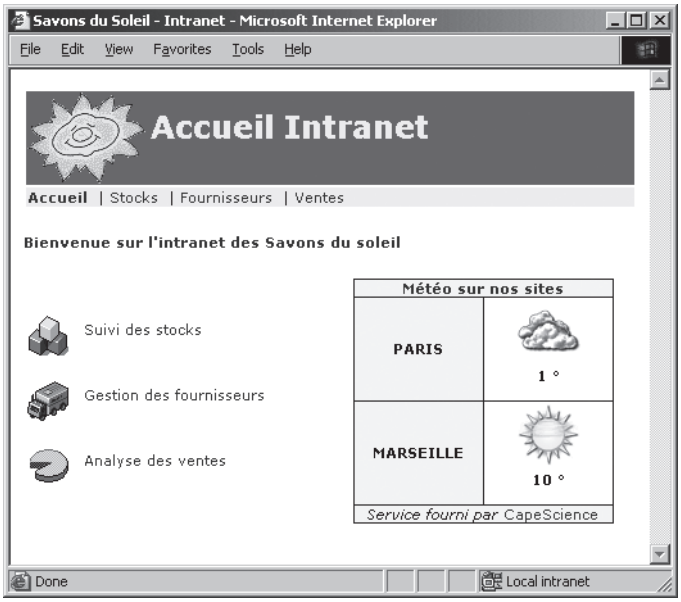



**Figure 8–12** Page d'accueil avec intégration du contrôle Meteo

# 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

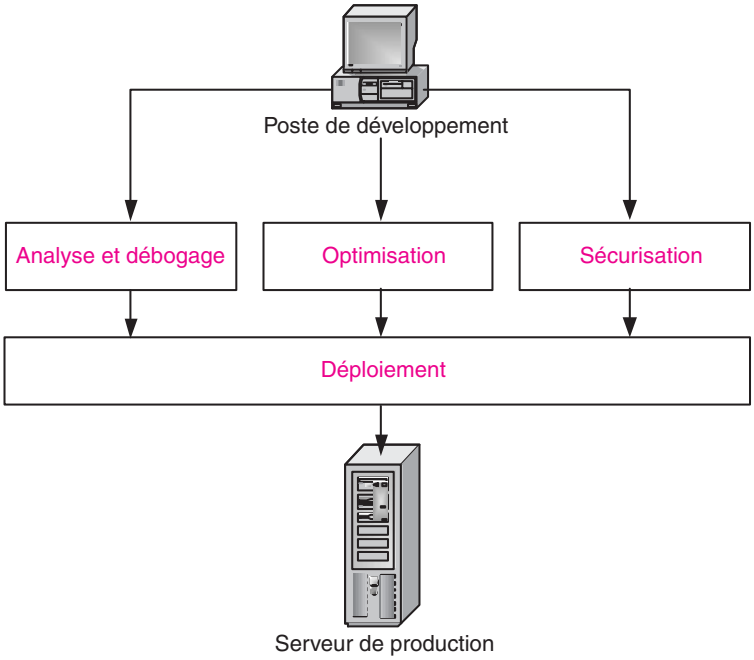
Authentification | machine.config | web.config | en-tête SOAP | débogueur .NET | Application_Error |

## SOMMAIRE


- Sécurisation d'une application ASP.NET
- Fichiers de configuration
- Débogage et gestion des erreurs
- Mécanisme de maintien en cache
- Déploiement vers un serveur de production


## 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).

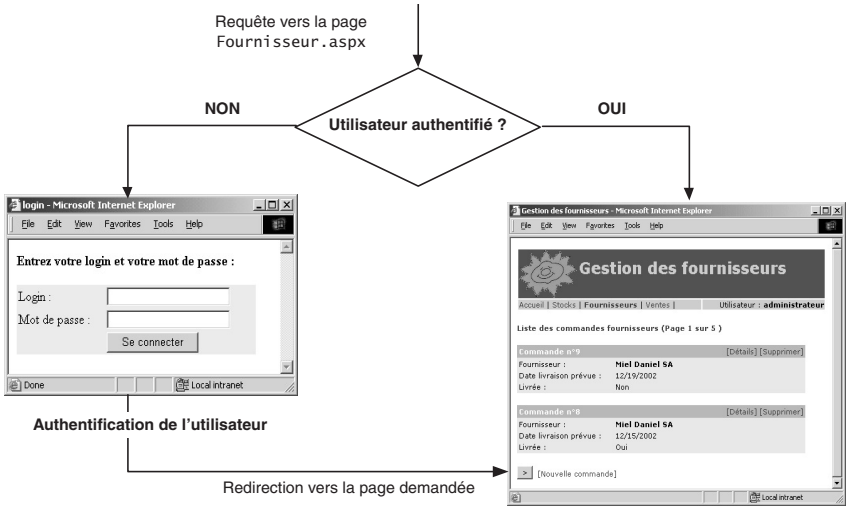



**Figure 9–1** Authentification des utilisateurs

**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 `HashPassword ForStoringInConfigFile`.

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 la 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.

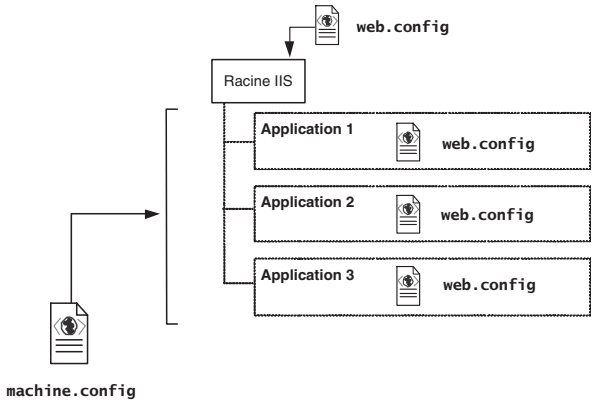



**Figure 9–2** Fichiers de configuration ASP.NET

#### 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)

```
< ?xml version= "1.0" encoding="UTF-8" ?>
<configuration>

  <system.web>
   ...
  </system.web>

  <appSettings>
     ...
  </appSettings>
</configuration>
```

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.).

**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.

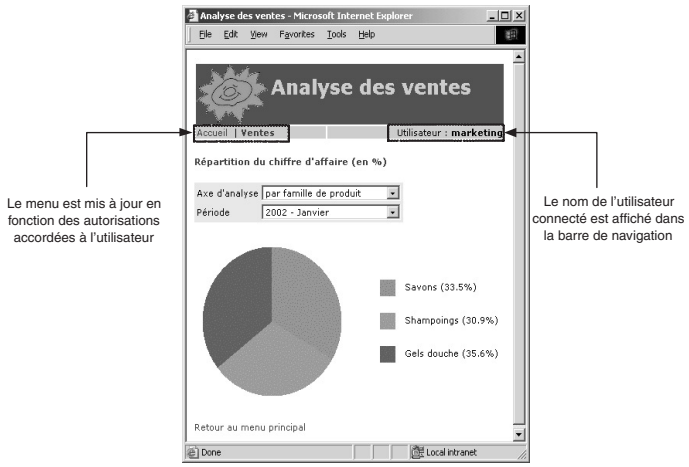



**Figure 9–3** Gestion sélective des autorisations

## 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.

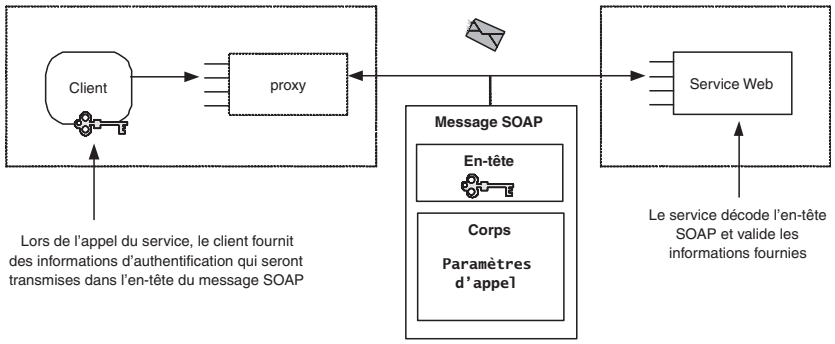



**Figure 9–4** Sécurisation de l'appel d'un service Web

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)**

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.

```
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
```

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).

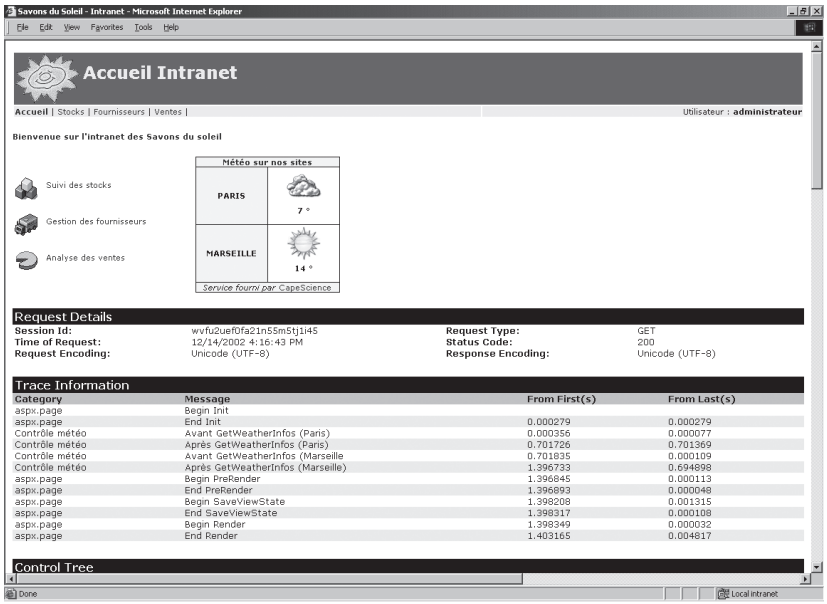



**Figure 9–5** Exécution d'une page avec activation de l'option Trace

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 `Performance Counter` 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.

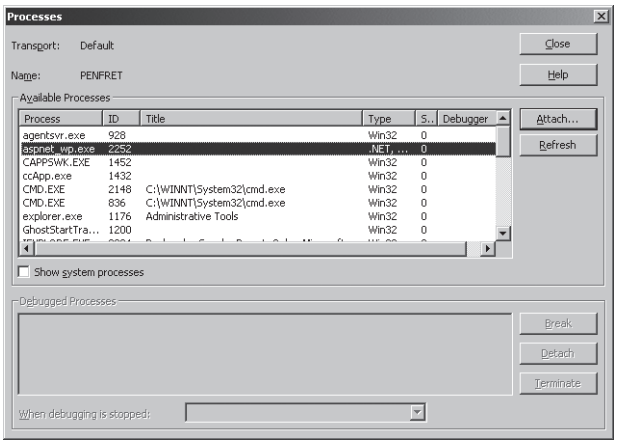



**Figure 9–6** Débogage du processus `aspnet_wp.exe`

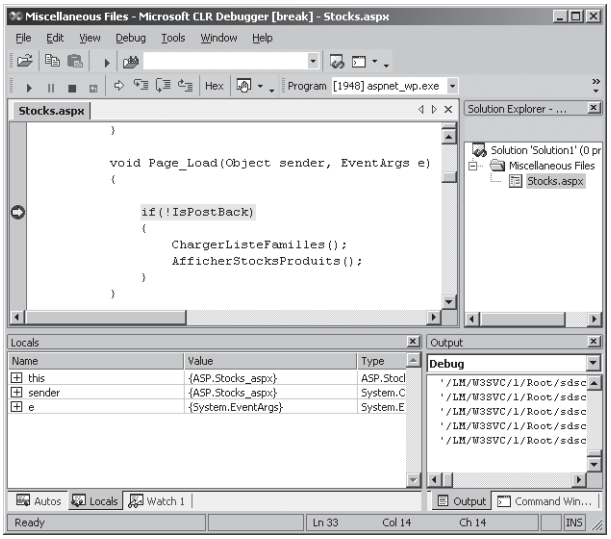



**Figure 9–7** Point d'arrêt dans le débogueur

**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

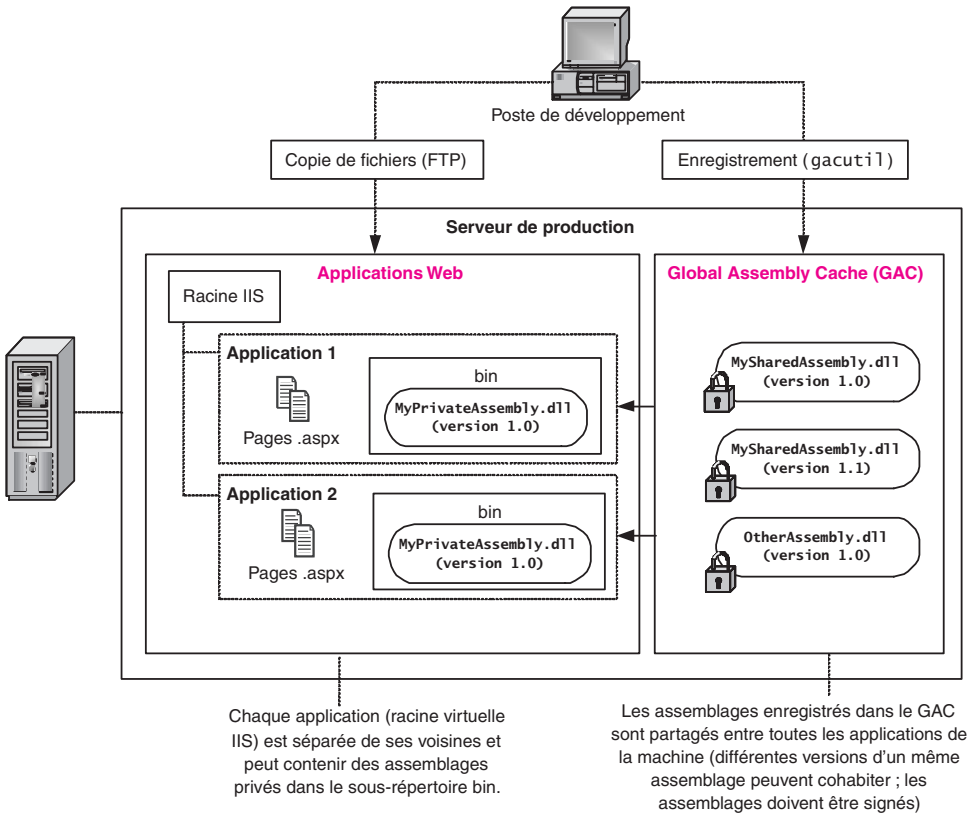



**Figure 9–9** Déploiement d'une application ASP.NET

### 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`).

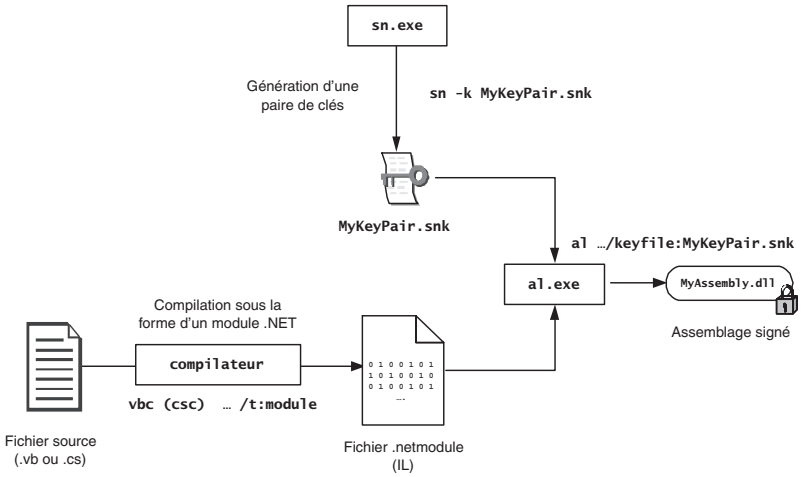



**Figure 9–10** Une des techniques possibles pour la signature d'un assemblage

### 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" />
```

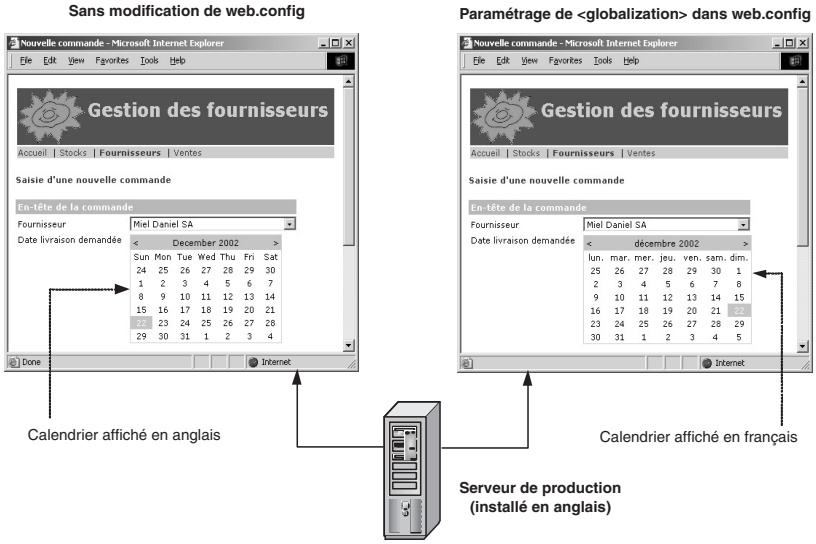



**Figure 9–11** Gestion de l'internationalisation

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>
```

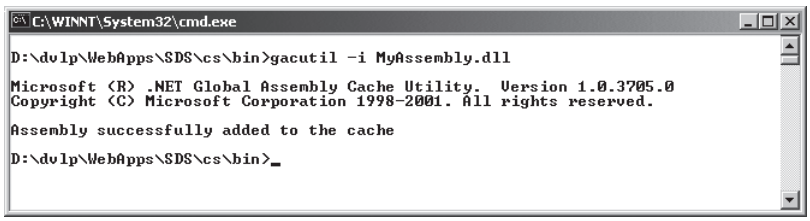



**Figure 9–12** Installation d'un assemblage dans le Global Assembly Cache

#### 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).

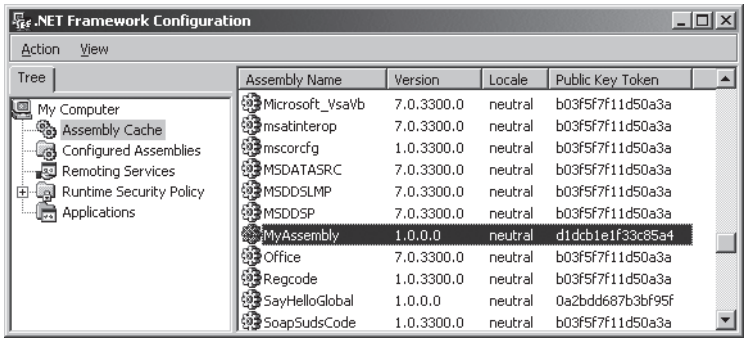



**Figure 9–13** Visualisation du contenu du Global Assembly Cache

# 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

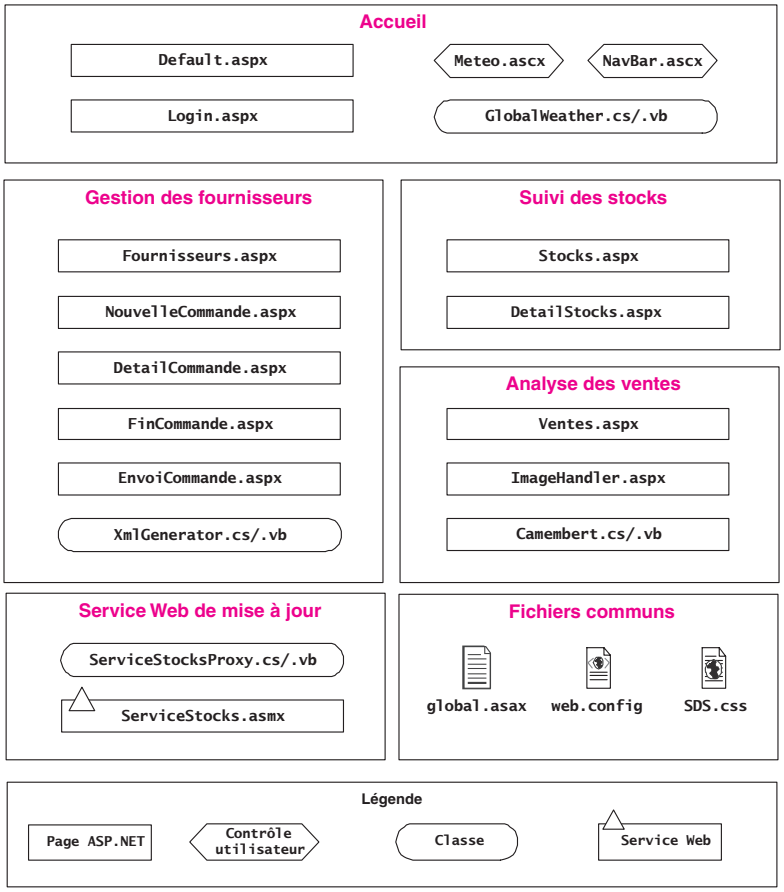


**Tableau A–1** Liste des fichiers de l'application

| 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

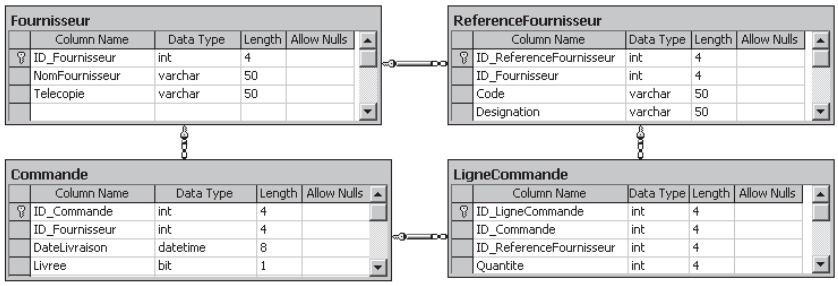



**Figure A–1** Schéma de la base (module fournisseur)

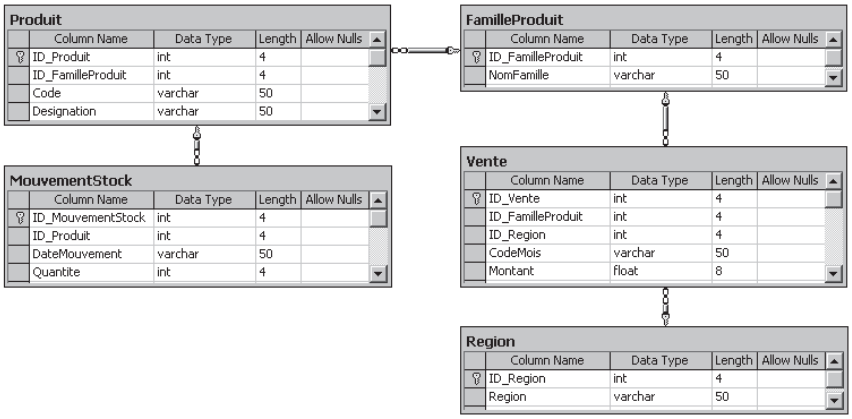



**Figure A–2** Schéma de la base (modules stocks et ventes)

**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`<br>`{`<br>`  get { ... return ... ;}`<br>`  set { ....= value; }`<br>`}` | `Public Property Name As String`<br>`  Get`<br>`  ...`<br>`  Return ...`<br>`  End Get`<br>`  Set`<br>`  ... = Value`<br>`  End Set`<br>`End Property` |
| Accéder à une propriété indexée | `s = Request.QueryString["id"];` | `s = Request.QueryString("id")` |
| Énumérer une collection | `foreach ( String s in coll )`<br>`{`<br>` ...`<br>`}` | `Dim s As String`<br>`For Each s In Coll`<br>` ...`<br>`Next` |
| Utiliser une boucle `for` | `for (int i=0; i<3; i++)`<br>`{`<br>` ...`<br>`}` | `Dim I As Integer`<br>`For I = 0 To 2`<br>`  ...`<br>`Next` |
| Utiliser une boucle `while` | `int i = 0;`<br>`while (i<3)`<br>`{`<br>`  ...`<br>`  i++;`<br>`}` | `Dim I As Integer`<br>`I = 0`<br>`Do While I < 3`<br>`  ...`<br>`  I += 1`<br>`Loop` |
| Utiliser un aiguillage `select/case` | `switch (i)`<br>`{`<br>`  case 1 :`<br>`  ...`<br>`  break;`<br><br>`  case 2 :`<br>`  ...`<br>`  break;`<br><br>`  default:`<br>`  ...`<br>`  break;`<br>`}` | `Select Case i`<br>`  Case 1`<br>`  ...`<br>`  Case 2`<br>`  ...`<br>`  Case Else`<br>`  ...`<br>`End Select` |

| Opération | C# | VB.NET |
|---|---|---|
| Utiliser un aiguillage `if/else` | `if (a == b)`<br>`{`<br>`  ...`<br>`}`<br>`else`<br>`{`<br>`  ...`<br>`}` | `If (a = b) Then`<br>`  ...`<br>`Else`<br>`  ...`<br>`End If` |
| Déclarer une méthode | `void f()`<br>`{`<br>`  ...`<br>`}`<br><br>`int g(int a)`<br>`{`<br>`  ...`<br>`  return i;`<br>`}` | `Sub f()`<br>`  ...`<br>`End Sub`<br><br>`Function g(a As Integer) _`<br>`As Integer`<br>`  ...`<br>`  Return i`<br>`End Function` |
| Gérer les exceptions | `try`<br>`{`<br>`  ...`<br>`}`<br>`catch(Exception e)`<br>`{`<br>`  ...`<br>`}`<br>`finally`<br>`{`<br>`  ...`<br>`}` | `Try`<br>`  ...`<br>`Catch e As Exception`<br>`  ...`<br>`Finally`<br>`  ...`<br>`End Try` |
| Effectuer une conversion de type | `c =(SqlConnection)Session["c"]` | `c=CType(Session("c"),_`<br>`  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`<br>`{`<br>`  public class C : Base`<br>`  {`<br>`  public C()`<br>`  {`<br>`  ... // Constructeur`<br>`  }`<br>`  ...`<br>`  }`<br>`}` | `Namespace N`<br><br>`Public Class C : Inherits Base`<br>`  Public Sub New()`<br>`  ... 'Constructeur`<br>`  End Sub`<br>`  ...`<br>`End Class`<br><br>`End Namespace` |
| Compiler une classe | `csc /out:Output.dll /t:library`<br>`➥ MyClass.cs /r:[references]` | `vbc /out:Output.dll /t:library`<br>`➥ 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


## Symboles

.ascx (fichier) 37, 39
.asmx (fichier) 155
.aspx (fichier) 33
.NET Framework (versions du) 20
.NET Redistributable 18
.NET SDK (voir kit de développement .NET) 18
<appSettings> (section) 172
<authentification> (section) 170
<compilation> (section) 60
<configuration> (section) 172
<credentials> (section) 171
<customErrors> (section) 179
<globalization> (section) 183
<system.web> (section) 172

## A

ADO.NET 54
al (utilitaire) 183
align (attribut) 41
AlternatingItemTemplate (contrôle Repeater) 84
application ASP.NET
  analyse 175
  configuration 172
  déboguage 177
  déploiement 181
  gestion des langues 183
  hébergement 181
  optimisation 178
  sécurisation 170
Application_Error (gestionnaire) 179
ApplicationException (classe) 179
ArrayList (classe) 64, 142
ASP, compatibilité avec 14
ASP.NET
  architecture 11
  filtre ISAPI 11
  processus principal 11
aspnet_filter.dll 11
aspnet_wp.exe 11
assemblage 36
  cohabitation de versions 184
  déploiement dans le GAC 184
  manifest 123
  ordre de recherche 184
  partagé 123, 182
  privé 123, 182
  signature 141, 182
  temporaire 123
Assembly (paramètre) 141
Assembly Linker 183
Attributes (propriété) 94
Authenticate (méthode) 171
authentification 170
AutoGenerateColumns (propriété) 69
AutoPostBack (propriété) 76, 105, 137
autorisations (gestion des) 171

## B

BeginTransaction (méthode) 112
Bibliothèque de classes .NET 18
Bitmap (classe) 148
BoundColumn 69
Button (contrôle serveur) 97

## C

cache (mécanisme de) 178
Cache (objet ASP.NET) 172
Calendar (contrôle serveur) 103
Catch 178
CausesValidation (propriété) 112
Click (événement) 93, 98
CLR, *voir* Common Language Runtime
code behind 16, 33, 35
code in-line 16
CommandArgument (propriété) 91
CommandType (propriété) 66
Commit (méthode) 112
Common Language Runtime (CLR) 12, 18
CompareValidator 113
compilateur
  C# 124, 147
  VB.NET 124, 147
compilateurs 12
compilation 47, 123
ConfigurationSettings 59
connexion (chaîne de) 58
Container (contrôle) 85
Context (propriété) 148
Control (directive) 37
contrôle serveur 32
  Button 97
  Calendar 103
  CompareValidator 113
  CustomValidator 113
  DataGrid 32, 50, 68
  DataList 84
  DropDownList 32, 50, 103, 137
  HyperLink 40, 90, 118
  Image 118, 165
  Label 40, 165
  LinkButton 86, 90
  RadioButtonList 97
  Range Validator 113
  RegularExpressionValidator 113, 130
  Repeater 81
  RequiredFieldValidator 113
  spécifique 136
  ValidationSummary 113
contrôle utilisateur 37, 39, 164
CssClass 44
CustomValidator (contrôle serveur) 113

## D

DataBind (méthode) 63
DataBinder (classe) 85
data-binding 86
DataGrid (contrôle serveur) 32, 50, 68
DataItem (propriété) 85
DataKeyField (propriété) 108
DataKeys (propriété) 108
DataList (contrôle serveur) 84
DataRowView (classe) 85
DataSet (classe) 106, 126
DataSource (propriété) 61, 83
DataTable (classe) 63, 156
DataTextField (propriété) 61
DataValueField (propriété) 61

DataView (classe) 63, 142
débogueur 176
Debug (mode) 60
Debug (option) 177
DeleteCommand (propriété) 99
Delphi 7 Studio 15
déploiement 181
documentation (.NET) 56
données (liaison de) 86
DrawString (méthode) 148
DropDownList (contrôle serveur) 32, 50, 103, 137
Duration (attribut) 178

**E**

EnableClientScript (propriété) 113
EnableSession (propriété) 156
EnableViewState (propriété) 76
erreur (gestion spécifique d'une) 179
ErrorMessage (propriété) 113
espace de nommage
- System.Diagnostics 176
- System.Drawing 146
- System.Web.Mail 130
- System.Web.Services 155
- System.Web.Services.Protocols 159
- System.Web.UI.WebControls 136
- System.Xml 127
- System.Xml.Xsl 127

événement (gestionnaire d') 34, 73
événements (remontée en bulle) 91
event bubbling 91
EventArgs 74
EVENTARGUMENT (contrôle caché) 76
EVENTTARGET (contrôle caché) 76
Exception (classe) 179
exceptions (gestion des) 178
ExecuteReader(méthode) 62
ExecuteXmlReader (méthode) 127
Explicit (Option) 121
expressions régulières 111

**F**

feuille de style 43
Fill (méthode) 99
FillPie (méthode) 148
FillRectangle (méthode) 148
Finally 179
FindControl (méthode) 94, 109
FooterTemplate (contrôle Repeater) 84
Format (méthode) 88
formatage 87, 88
Forms (type d'authentification) 170
FormsAuthentication (classe) 171
formulaire Web 13
FromImage (méthode) 148

**G**

GAC, *voir* Global Assembly Cache
gacutil (utilitaire) 183
Global Assembly Cache (GAC) 123, 139, 182
global.asax (fichier) 55, 180
Graphics (classe) 148

**H**

HashTable 64
HeaderTemplate (contrôle Repeater) 84
hébergement 181
héritage 144
HtmlTextWriter (classe) 145
HyperLink (contrôle serveur) 40, 86, 90, 118
HyperLinkColumn 69

**I**

ICollection (interface) 64
IIS, *voir* Internet Information Server
IL, *voir* Intermediate Language
ILDASM (utilitaire) 47
Image (contrôle serveur) 118, 165
ImageUrl (méthode) 118
Import (directive) 59, 65
Inherits (attribut) 36
InsertCommand (propriété) 99
Intermediate Language 12, 124
internationalisation 183
Internet Information Server (IIS) 19, 48
Invoke (méthode) 160
IsPostBack (propriété) 75, 141
ItemTemplate (contrôle Repeater) 84

**J**

Just-In-Time (compilateur) 36

**K**

kit de développement .NET
- installation 20
- prérequis pour l'installation 19
- téléchargement 18

**L**

Label (contrôle serveur) 40, 165
langue (configuration de la) 183
LinkButton (contrôle serveur) 86, 90
loginUrl (attribut) 170

**M**

machine.config (fichier) 172
MailAttachement (classe) 130
MailMessage (classe) 130
MapPath (méthode) 125
message (envoi de) 130
Microsoft Access 27
Microsoft Data Access Components (MDAC) 19
MSDE (base de données) 14, 18
MSDN 57
MSIL, *voir* Intermediate Language

**N**

NavigateUrl (propriété) 90

**O**

OBDC 55
objet métier 119
- compilation 123

OLE-DB 55
OleDbConnection 58
onclick (événement client) 93
OnItemCommand (événement) 91
OnItemCreated (événement) 93
optimisation 178
osql (exécutable) 27
OutputParam (propriété) 109
override 147

**P**

Page (classe) 95
Page (directive) 34
Page_Init (événement) 73
Page_Load (événement) 35, 73
Page_Unload (événement) 73
PagedDataSource (classe) 95
pagelet 37
pagination (mécanisme de) 95
PerformanceCounter (classe) 176
proxy 160
PublicKeyToken (paramètre) 141

**Q**

Quick Tag Edit (Web Matrix) 97

## R

RadioButtonList (contrôle serveur) 97
RangeValidator (contrôle serveur) 113
RedirectFromLoginPage (méthode) 171
Reflector 57
Register (directive) 37, 140
RegularExpressionValidator (contrôle serveur) 113, 130
Render (méthode) 144
Repeater (contrôle serveur) 81
Request (objet) 71, 125
RequiredFieldValidator (contrôle serveur) 113
Response (objet) 150
Rollback (méthode) 112

## S

SelectCommand (propriété) 99
SelectedIndexChanged(événement) 137
Send (méthode) 132
SeparatorTemplate (contrôle Repeater) 84
service Web 153
  description 162
  point d'accès 162
  protocoles 159
  proxy 160
  sécurisation 173
  types utilisables 155
Session (objet) 55, 95, 156
signature numérique 182
SMTP, serveur 130
sn (utilitaire) 182
SOAP 159, 160
SoapDocumentMethodAttribute (attribut) 161
SoapHeader (classe) 174
SoapHttpClientProtocol (classe) 159
SQL Server Enterprise Manager 26
SqlCommand (classe) 62, 137
SqlCommandBuilder (classe) 107, 157
SqlConnection (classe) 55, 63
SqlDataAdapter (classe) 63, 99, 126, 156
  ajout de données 106
  mise à jour de données 100
SqlDataReader (classe) 62, 137
SqlTransaction (classe) 112
Src (attribut) 36
StmpMail (classe) 130
Strict (Option) 121
String (classe) 88
System.Data 54
System.Data.OleDb 54
System.Data.SqlClient 54
System.Diagnostics 176
System.Drawing 146
System.Web.Mail 130
System.Web.Services 155
System.Web.Services.Protocols 159
System.Web.UI 32
System.Web.UI.WebControls 136
System.Xml 127
System.Xml.Xsl 127

## T

TagName (attribut) 45
TagPrefix (attribut) 45
TemplateColumn (contrôle DataGrid) 103
trace (mécanisme de) 175
Trace (Option) 175
Trace (propriété) 175
transaction (gestion de) 112
Transaction (propriété) 112
Try 178

## U

UDDI 161
Update (méthode) 99
UpdateCommand (propriété) 99
URI 159
User (propriété) 171

## V

validation (contrôles de), *voir* contrôle serveurs
ValidationSummary (contrôle serveur) 113
VaryByCustom (attribut) 178
VaryByHeader (attribut) 178
VaryByParam (attribut) 178
Version (paramètre) 141
VIEWSTATE (contrôle caché) 73
ViewState (propriété) 95, 147
Visual Studio.NET 15
VisualMap (contrôle serveur spécifique) 136

## W

Web Matrix 15, 38
  connexion à une base existante 23
  création d'une base 23
  création d'une procédure stockée 25
  création d'une table 24
  installation 22
  interface de gestion des données 23
web.config (fichier) 60, 88, 172
WebControl (classe) 144
WebMethod (attribut) 155
WebService (classe) 156
WebService (directive) 155
WebServiceBindingAttribute (attribut) 160
WriteXml (méthode) 126
WriteXmlSchema (méthode) 126
WSDL 158
wsdl.exe (utilitaire) 161

## X

XML
  génération de documents 126
  schéma 126
XmlDocument (classe) 127
XmlNodeList 64
XmlTextReader (classe) 127
XmlTextWriter (classe) 127
XmlValidatingReader (classe) 129
XPathDocument (classe) 129
XSL
  feuille de style 127
  transformations 127
XslTransform (classe) 128