12

Bases de données

ADO.NET est le successeur d'ADO (*ActiveX Data Objects*). Ses fonctionnalités permettent d'accéder simplement à des bases de données locales ou distantes. À travers un ensemble de procédures et de fonctions, il est possible de visualiser, de modifier et de supprimer les données contenues dans une base de données, que celle-ci se trouve sur l'ordinateur local ou sur un ordinateur distant.

Pour illustrer le fonctionnement d'ADO.NET, nous allons prendre deux exemples. Le premier permettra d'accéder à une base de données locale Microsoft Access. Le second sera une copie conforme du premier, à ceci près que les données se trouveront sur un serveur distant.

Accès à une base de données locale

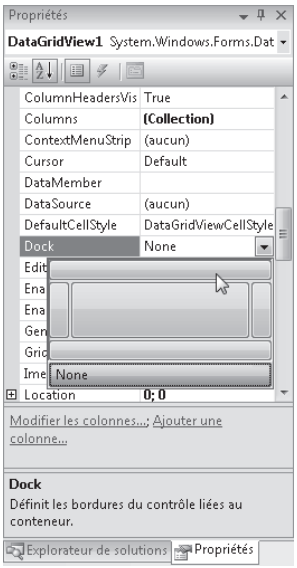
Lancez la commande Nouveau/Projet dans le menu Fichier. Choisissez .NET Framework 4 dans la liste déroulante Framework. Sélectionnez Visual Basic/Windows dans le volet gauche et Application Windows Forms dans le volet central. Nommez le projet db et validez en cliquant sur OK. Ajoutez un contrôle `DataGridView` et quatre boutons de commande

à la feuille du projet et modifiez les propriétés de ces objets comme indiqué dans le tableau suivant :

Contrôle	*Propriété*	*Valeur*
`Form1`	`Text`	Accès à une base locale
`Button1`	`Text`	Bibliothèque
`Button1`	`Name`	Bibliothèque
`Button2`	`Text`	Tous
`Button2`	`Name`	Tous
`Button3`	`Text`	Mise à jour
`Button3`	`Name`	MiseAJour
`Button4`	`Text`	Quitter
`Button4`	`Name`	Quitter

Pour terminer la mise en forme de l'application, cliquez sur le contrôle `DataGridView` et affectez la valeur `Top` à sa propriété `Dock`. Pour cela, développez cette propriété en déroulant la liste `Dock` et cliquez sur le rectangle supérieur (voir Figure 12.1).

Figure 12.1
Affectation de la valeur `Top` *à la propriété* `Dock` *du contrôle* `DataGridView1`.



La feuille de l'application devrait maintenant avoir l'allure de la Figure 12.2.

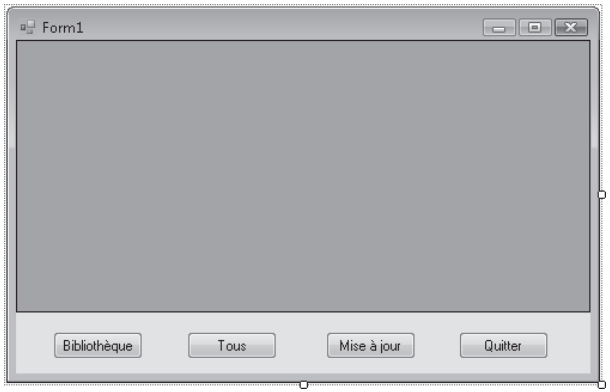


Figure 12.2
L'application db mise en forme, en mode Édition.

Cette application utilise les bibliothèques `System.Data` et `System.Data.OleDB`. Ajoutez les deux instructions ci-après en tête de listing :

```
Imports System.Data
Imports System.Data.OleDb
```

Définissez les objets utilisés par l'application à la suite du code généré par Windows Form Designer :

```
Dim Connexion As String
Dim ConnexionOLE As OleDBConnection = New oledbConnection()
Dim da As OleDbDataAdapter
Dim ds As DataSet
Dim dv As DataView
Dim cb As OleDbCommandBuilder
```

L'exemple choisi repose sur une base de données Access librement téléchargeable sur la page **http://www.info-3000.com/access/bestofgestion/index.php**.

Cette base contient plusieurs tables de données. Nous allons nous intéresser à l'une d'entre elles, qui a pour nom "Catégories". Pour afficher l'ensemble des enregistrements qui la composent, nous allons définir la procédure `AfficheTous()` :

```
Public Sub AfficheTous()
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
    ➥C:\\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    ConnexionOLE.Open()
    da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
    ds = New DataSet()
    da.Fill(ds, "mytable")
```

```
        dv = ds.Tables("mytable").DefaultView
        ConnexionOLE.Close()
        DataGridView1.DataSource = dv
        dv.AllowEdit = True
    End Sub
```

La première ligne initialise la chaîne `Connexion` avec une valeur qui indique le type (`Provider`) et le nom (`Data Source`) de la base de données à laquelle accéder :

```
Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\bestofgestion97.mdb"
```

Remarquez le séparateur ";" entre les deux paramètres de la chaîne Connexion. Si nécessaire, vous utiliserez ce même séparateur pour préciser le nom d'utilisateur (User ID) et le mot de passe (password).

Cette chaîne est passée à l'objet `OleDBConnexion` `ConnexionOLE` pour établir la connexion avec la base de données :

```
ConnexionOLE.ConnectionString = Connexion
```

Puis un objet `OleDbDataAdapter` est utilisé pour accéder à la table `T_TravailObjet` de la base :

```
da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
```

Les données de la table `T_TravailObjet` sont placées dans un objet `DataSet`. Ce dernier est rempli à l'aide de la méthode `Fill()` :

```
ds = New DataSet()
da.Fill(ds, "mytable")
```

Pour que l'objet `DataSet` puisse être édité par l'application, il est nécessaire de le copier dans un objet `DataView` :

```
dv = ds.Tables("mytable").DefaultView
```

Il suffit maintenant de l'associer au contrôle `DataGridView` et de valider son édition :

```
DataGridView1.DataSource = dv
dv.AllowEdit = True
```

Pour que tous les enregistrements de la table Catégories soient visualisés dans le contrôle `DataGrid` dès l'exécution de l'application, il suffit d'invoquer la procédure `AfficheTous()` dans `Form1_Load()`. Double-cliquez sur un emplacement inoccupé de la feuille et complétez la procédure `Form1_Load()` comme suit :

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles MyBase.Load
    AfficheTous()
End Sub
```

La fenêtre de l'application se présente alors comme montré à la Figure 12.3.

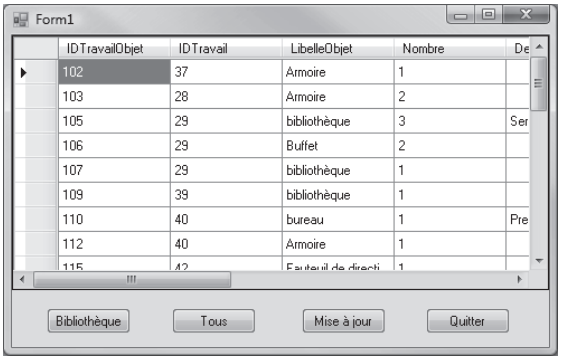


Figure 12.3
Affichage de tous les enregistrements de la table `T_TravailObjet`.

Nous allons maintenant associer du code aux quatre boutons de commande de la feuille. Le bouton Bibliothèque va limiter l'affichage aux enregistrements dont le champ `LibelleObjet` vaut `"bibliothèque"`. Double-cliquez sur le premier bouton et complétez la procédure `Bibliothèque_Click()` comme suit :

```
Private Sub Bibliothèque_Click(ByVal sender As System.Object, ByVal e As System.
➥EventArgs) Handles Condiments.Click
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
    ➥C:\\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    da = New OleDbDataAdapter("Select * from T_TravailObjet where [LibelleObjet]
    ='bibliothèque'", ConnexionOLE)
    ds = New DataSet()
    da.Fill(ds, "mytable")
    dv = ds.Tables("mytable").DefaultView
    DataGrid1.DataSource = dv
    dv.AllowEdit = True
End Sub
```

Cette procédure est très proche de `AfficheTous()`, à ceci près que la commande d'initialisation de l'objet `OleDbDataAdapter` utilise une clause `where` pour que seuls les enregistrements dont le champ `Nom de catégorie` vaut `Condiments` soient sélectionnés :

```
da = New OleDbDataAdapter("Select * from T_TravailObjet where [LibelleObjet]
➥='bibliothèque'", ConnexionOLE)
```

Double-cliquez sur le bouton Tous. La procédure `Tous_Click()` est identique à la procédure `Form1_Load()`. Elle se contente d'invoquer `AfficheTous()` afin d'afficher tous les enregistrements de la table Catégories :

```
Private Sub Tous_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Tous.Click
    AfficheTous()
End Sub
```

Les champs de données de la table peuvent être modifiés. Mais, attention, si vous refermez l'application, ils ne seront pas automatiquement sauvegardés dans la base de données ! Pour cela, vous devrez appeler la méthode `Update()`. Cette action sera effectuée en appuyant sur le bouton Mise à jour.

Double-cliquez sur ce bouton et complétez la procédure `MiseAJour_Click()` comme suit :

```
Private Sub MiseAJour_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles MiseAJour.Click
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
➥C:\\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    cb = New OleDbCommandBuilder(da)
    da.Update(ds, "mytable")
End Sub
```

Après avoir spécifié la base de données à utiliser :

```
Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\bestofgestion97.mdb"
ConnexionOLE.ConnectionString = Connexion
```

l'objet `OleDbCommandBuilder` `cb` est créé a partir de l'objet `OleDbDataAdapter` `da` :

```
cb = New OleDbCommandBuilder(da)
```

Il suffit maintenant d'utiliser la méthode `Update()` pour mettre à jour la table avec le contenu de l'objet `OleDbDataAdapter` :

```
da.Update(ds, "mytable")
```

Pour terminer, double-cliquez sur le bouton Quitter et ajoutez une instruction `End` dans la procédure `Quitter_Click()` :

```
Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Quitter.Click
    End
End Sub
```

Voici le listing complet de l'application. Les fichiers correspondants se trouvent dans le dossier db après installation des sources de l'ouvrage.

```
Imports System.Data
Imports System.Data.OleDb
Public Class Form1
    Dim Connexion As String
    Dim ConnexionOLE As OleDbConnection
    Dim da As OleDbDataAdapter
    Dim ds As DataSet
    Dim dv As DataView
    Dim cb As OleDbCommandBuilder

    Public Sub AfficheTous()
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥C:\\bestofgestion97.mdb"
        ConnexionOLE = New OleDbConnection
        ConnexionOLE.ConnectionString = Connexion
        da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
        ds = New DataSet()
        da.Fill(ds, "mytable")
        dv = ds.Tables("mytable").DefaultView
        DataGridView1.DataSource = dv
        dv.AllowEdit = True
    End Sub



    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles MyBase.Load
        AfficheTous()
    End Sub

    Private Sub Bibliothèque_Click(ByVal sender As System.Object, ByVal e
    ➥As System.EventArgs) Handles Bibliothèque.Click
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥C:\\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
        da = New OleDbDataAdapter("Select * from T_TravailObjet where [LibelleObjet]
        ➥='bibliothèque'", ConnexionOLE)
        ds = New DataSet()
        da.Fill(ds, "mytable")
        dv = ds.Tables("mytable").DefaultView
        DataGridView1.DataSource = dv
        dv.AllowEdit = True
    End Sub
```

```
    Private Sub Tous_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Tous.Click
        AfficheTous()
    End Sub

    Private Sub MiseAjour_Click(ByVal sender As System.Object, ByVal e As
    ➥System.EventArgs) Handles MiseAjour.Click
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥C:\\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
        cb = New OleDbCommandBuilder(da)
        da.Update(ds, "mytable")
    End Sub

    Private Sub Quitter_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Quitter.Click
        End
    End Sub
End Class
```

Accès à une base de données distante

Dans cette section, nous allons vous montrer comment créer une application capable d'exploiter des données contenues dans une base de données distante. Cette base a pour nom BestOfGestion.mdb. Elle est librement téléchargeable sur la page **www.info-3000.com/access/bestofgestion/index.php**. La technique utilisée relève des services web. Pour tous renseignements complémentaires, consultez le Chapitre 25.

Définition du service web

Lancez la commande Nouveau/Site web du menu Fichier. Sélectionnez .NET Framework 3.5 dans la liste déroulante Frameworks. Sélectionnez le modèle Service web ASP.NET, donnez le nom `Service1` au nouveau service web et validez en cliquant sur OK.

Ajoutez les clauses `Imports` suivantes en tête de listing :

```
Imports System.Data
Imports System.Data.OleDb
```

La clause `<webservice>` (devant le mot-clé `Public`) indique que la classe `Service1` sera utilisée comme service web :

```
<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService
```

Lorsque, par la suite, le service sera implanté sur un serveur web, vous devrez préciser son nom de domaine à la suite du mot-clé `NameSpace`.

Définissez les variables utilisées par le service web :

```
Dim Connexion As String
Dim ConnexionOLE As OleDbConnection = New OleDbConnection()
Dim da As OleDbDataAdapter
Dim ds As DataSet
```

Supprimez la fonction `HelloWorld()`, générée automatiquement par Visual Studio, et remplacez-la par la fonction `AfficheTous()` suivante :

```
<WebMethod()> Public Function AfficheTous() As DataSet
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
    ➥c:\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
    ds = New DataSet()
    da.Fill(ds, "mytable")
    Return ds
End Function
```

Cette fonction sera exploitée sur le Web (`<WebMethod()>`). Elle ne demande aucun argument ; mais renvoie un `DataSet` :

```
<WebMethod()> Public Function AfficheTous() As DataSet
```

Les deux lignes suivantes établissent une connexion avec la base de données. Ici, il s'agit de la base Access c:\bestofgestion97.mdb :

```
Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\bestofgestion97.mdb"
ConnexionOLE.ConnectionString = Connexion
```

Tous les enregistrements de la table `T_TravailObjet` sont sélectionnés et placés dans l'objet `OleDbDataAdapter da` :

```
da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
```

Puis le DataSet ds est rempli avec l'objet da :

```
ds = New DataSet()
da.Fill(ds, "mytable")
```

Enfin, l'objet DataSet ds est retourné par la fonction :

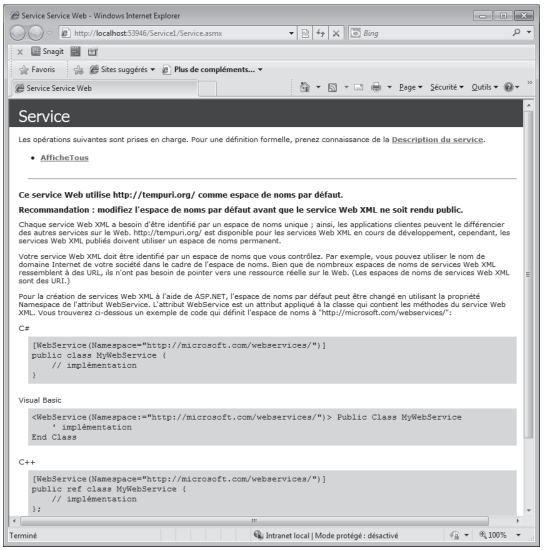
```
Return ds
```

Voici le code complet de la classe Service1 :

```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.OleDb


<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService

    Dim Connexion As String
    Dim ConnexionOLE As OleDbConnection = New OleDbConnection()
    Dim da As OleDbDataAdapter
    Dim ds As DataSet

    <WebMethod()> Public Function AfficheTous() As DataSet
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥c:\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
        da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
        ds = New DataSet()
        da.Fill(ds, "mytable")
        Return ds
    End Function

End Class
```

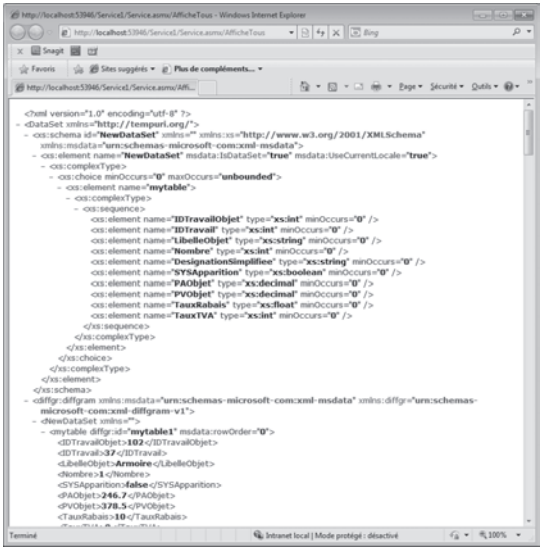
Appuyez sur la touche F5. La fonction AfficheTous() peut être testée (voir Figure 12.4).

Figure 12.4
Cette page permet de tester la fonction `AfficheTous`.



Cliquez sur le lien AfficheTous puis sur le bouton Appeler. Le résultat de la fonction `AfficheTous()` apparaît sous une forme XML (voir Figure 12.5).

Figure 12.5
Résultat de la fonction `AfficheTous()`.



Dans la barre de notification, une icône indique que le service web est en cours d'exécution et précise le port utilisé (voir Figure 12.6).

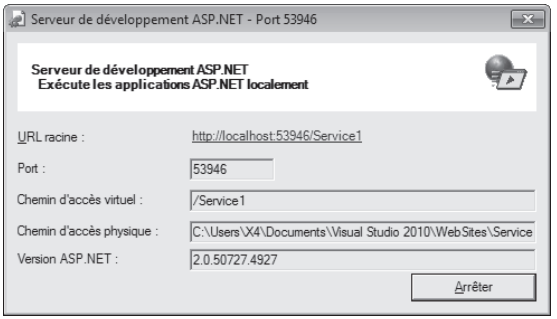


Figure 12.6
Dans cet exemple, le service web est accessible sur le port 53946.

Ne terminez pas l'exécution du service. Nous en aurons besoin dans la section suivante.

Utilisation du service web

Vous allez maintenant définir un projet qui exploite le service web défini dans la section précédente. Ouvrez une nouvelle instance de Visual Studio et lancez la commande Nouveau > Site web du menu Fichier. Choisissez .NET Framework 3.5 dans la première liste déroulante et sélectionnez le modèle Site web ASP.NET. Donnez le nom TestBDDistante à l'application et validez (cette application se trouve dans le dossier Projects\TestBDDistante des sources de l'ouvrage).

Cliquez du bouton droit sur l'entrée C:\…\TestBDDistante dans l'Explorateur de solutions et choisissez Ajouter une référence web dans le menu surgissant. Saisissez l'adresse correspondant au service web en cours de débogage dans la zone URL (voir Figure 12.6).

Pour connaître le port utilisé par le service en cours de débogage, il suffit de pointer l'icône Serveur de développement ASP.NET dans la barre de notification (voir Figure 12.7).

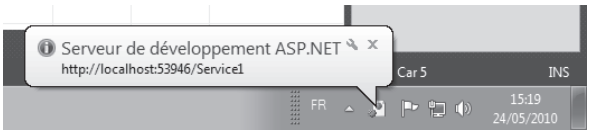
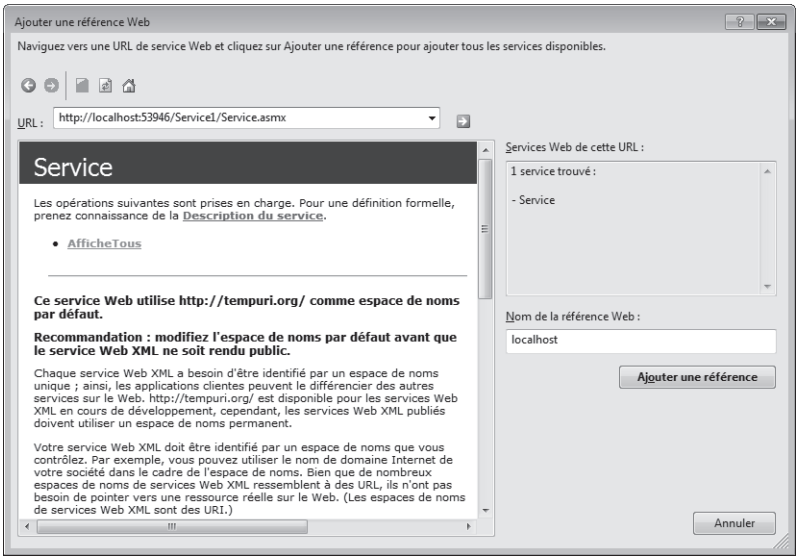


Figure 12.7
L'icône Serveur de développement ASP.NET indique le port utilisé par le service web.

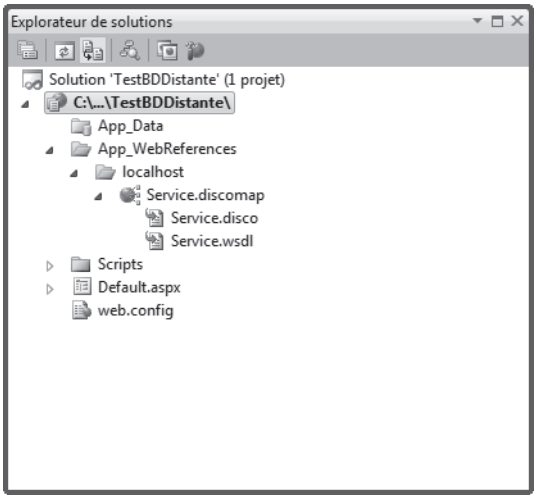
Cliquez sur le bouton Aller à. La boîte de dialogue Ajouter une référence web se présente comme montré à la Figure 12.8.

Figure 12.8
Le service `Service1` *est sur le point d'être intégré au projet web.*



Cliquez sur Ajouter une référence. Le service web `Service1` est maintenant référencé dans le projet. Vous pouvez donc utiliser sa méthode `AfficheTous()` (voir Figure 12.9).

Figure 12.9
Le service web `Service1` *est référencé dans le projet.*



Ajoutez un contrôle `GridView` sur la page de l'application (ce contrôle se trouve sous l'onglet Données de la Boîte à outils). La fenêtre de conception se présente comme montré à la Figure 12.10.

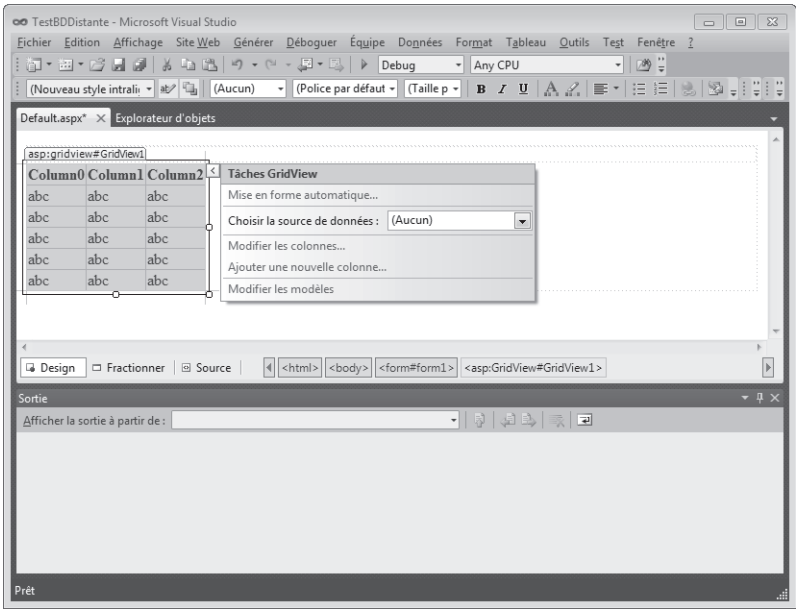


Figure 12.10
Le contrôle `GridView` *a été placé sur la feuille.*

Pour rendre ce contrôle opérationnel, vous devez l'initialiser. Déroulez la liste Choisir la source de données et sélectionnez l'entrée <Nouvelle source de données>. Une boîte de dialogue intitulée Assistant Configuration de source de données s'affiche (voir Figure 12.11).

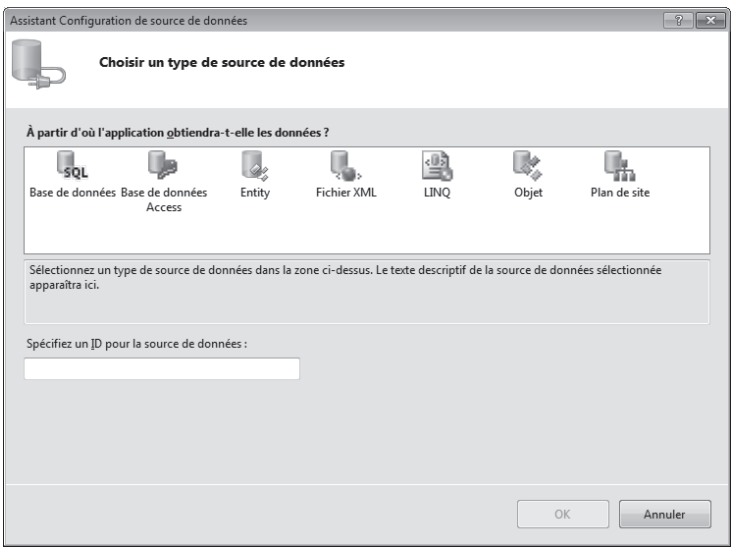
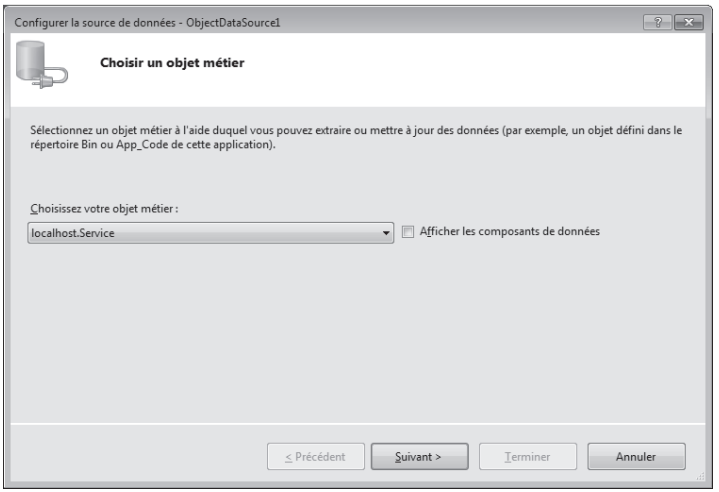


Figure 12.11
Cet assistant va vous aider à associer la fonction `AfficheTous()` *au contrôle* `GridView`.

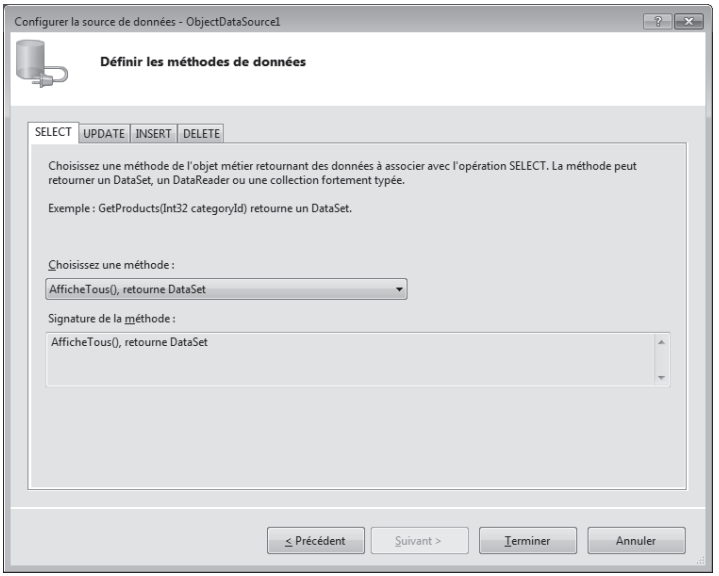
Sélectionnez Objet dans la zone À partir d'où l'application obtiendra-t-elle les données, puis validez en cliquant sur OK. L'assistant vous demande alors de choisir un "métier" par l'intermédiaire duquel vous pourrez mettre à jour les données de la table (voir Figure 12.12).

Figure 12.12
Choix du métier localhost.Service.



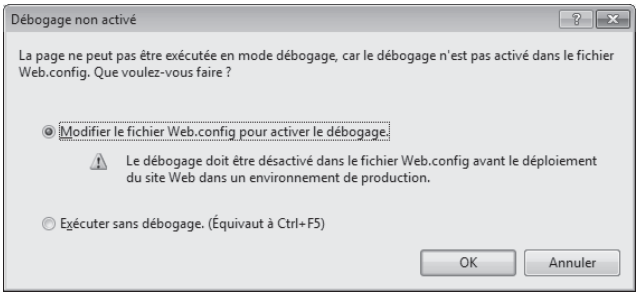
Choisissez localhost.Service et cliquez sur Suivant. Vous devez maintenant choisir la méthode qui permettra de remplir l'objet `GridView`. Déroulez la liste et choisissez la méthode `AfficheTous()` (voir Figure 12.13).

Figure 12.13
La méthode `AfficheTous()` *va permettre de remplir le* `GridView`.



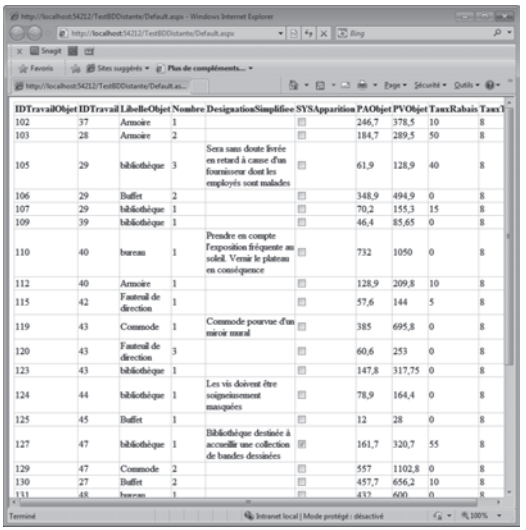
Cliquez sur Terminer. Vous pouvez vérifier que la méthode `AfficheTous()` du service web `Service1` est bien exécutée en cliquant sur le bouton Démarrer le débogage. Une boîte de dialogue vous informe que le débogage ne peut pas être réalisé (voir Figure 12.14). Sélectionnez l'option Modifier le fichier web.config pour activer le débogage et validez en cliquant sur OK.

Figure 12.14
Vous devez activer le débogage.



Au bout de quelques secondes, l'Explorateur web affiche le contenu de la table renvoyée par la méthode `AfficheTous()` (voir Figure 12.15).

Figure 12.15
La communication avec le service web a renvoyé les données escomptées.



IDTravailObjet	IDTravail	LibelleObjet	Nombre	DesignationSimplifiee	SYSApparition	PAObjet	PVObjet	TauxRabais	TauxT
102	37	Armoire	1			246,7	378,5	10	8
103	28	Armoire	2			184,7	289,5	50	8
105	29	bibliothèque	3	Sera sans doute livrée en retard à cause d'un fournisseur dont les employés sont malades		61,9	128,9	40	8
106	29	Buffet	2			348,9	494,9	0	8
107	29	bibliothèque	1			70,2	155,3	15	8
109	39	bibliothèque	1			46,4	85,65	0	8
110	40	bureau	1	Prendre en compte l'exposition fréquente au soleil. Vernir le plateau en conséquence		732	1050	0	8
112	40	Armoire	1			128,9	209,8	10	8
115	42	Fauteuil de direction	1			57,6	144	5	8
119	43	Commode	1	Commode pourvue d'un miroir mural		385	695,8	0	8
120	43	Fauteuil de direction	3			60,6	253	0	8
123	43	bibliothèque	1			147,8	317,75	0	8
124	44	bibliothèque	1	Les vis doivent être soigneusement masquées		78,9	164,4	0	8
125	45	Buffet	1			12	28	0	8
127	47	bibliothèque	1	Bibliothèque destinée à accueillir une collection de bandes dessinées	☑	161,7	320,7	55	8
129	47	Commode	2			557	1102,8	0	8
130	27	Buffet	2			457,7	656,2	10	8
131	48	bureau	1			432	600	0	8

Utilisation plus complète du service web

Nous allons maintenant améliorer le service web et l'application qui l'utilise pour adresser des commandes SQL à la base de données bestofgestion97.mdb.

Ouvrez le service web Service1. Ajoutez la méthode AfficheBiblio() pour ne sélectionner que les condiments dans la table Catégories :

```
<WebMethod()> Public Function AfficheBiblio() As DataSet
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
    ➥C:\\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    da = New OleDbDataAdapter("Select * from T_TravailObjet where [LibelleObjet]
    ➥='bibliothèque'", ConnexionOLE)
    ds = New DataSet()
    da.Fill(ds, "mytable")
    Return ds
End Function
```

Cette méthode est très proche de AfficheTous(), à ceci près que l'instruction de sélection des enregistrements dans la table T_TravailObjet est plus restrictive. Seuls les enregistrements dont le champ LibelleObjet a pour valeur bibliothèque sont sélectionnés :

```
da = New OleDbDataAdapter("Select * from T_TravailObjet where
[LibelleObjet]='bibliothèque'", ConnexionOLE)
```

Définissez maintenant la méthode AfficheSQL(). Elle permettra à l'utilisateur d'interroger la base de données en entrant une requête SQL de son choix :

```
<WebMethod()> Public Function AfficheSQL(ByVal st As String) As DataSet
    Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
    ➥C:\\bestofgestion97.mdb"
    ConnexionOLE.ConnectionString = Connexion
    da = New OleDbDataAdapter(st, ConnexionOLE)
    ds = New DataSet()
    da.Fill(ds, "mytable")
    Return ds
End Function
```

Remarquez la définition de la fonction AfficheSQL(). Celle-ci admet un argument String et retourne un DataSet :

```
<WebMethod()> Public Function AfficheSQL(ByVal st As String) As DataSet
```

La chaîne st passée en argument de la fonction est utilisée pour sélectionner les enregistrements dans l'objet da :

```
da = New OleDbDataAdapter(st, ConnexionOLE)
```

Voici le code complet du service web Service1. Ce code, ainsi que les fichiers complémentaires au projet, se trouvent dans le dossier Service1 après installation des sources de l'ouvrage.

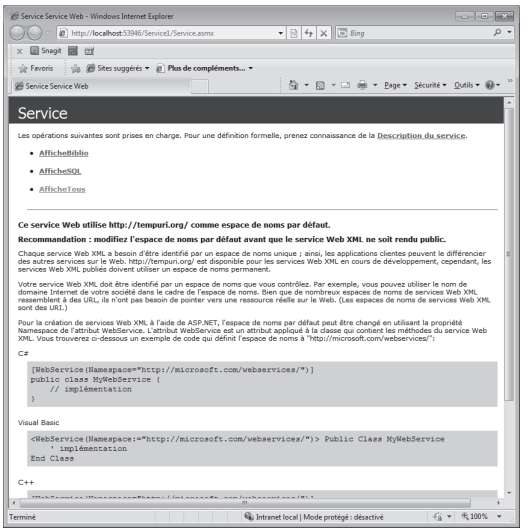
```
Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Data
Imports System.Data.OleDb


<WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Public Class Service
     Inherits System.Web.Services.WebService

    Dim Connexion As String
    Dim ConnexionOLE As OleDbConnection = New OleDbConnection()
    Dim da As OleDbDataAdapter
    Dim ds As DataSet

    <WebMethod()> Public Function AfficheTous() As DataSet
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥c:\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
        da = New OleDbDataAdapter("Select * from T_TravailObjet", ConnexionOLE)
        ds = New DataSet()
        da.Fill(ds, "mytable")
        Return ds
    End Function

    <WebMethod()> Public Function AfficheBiblio() As DataSet
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥C:\\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
        da = New OleDbDataAdapter("Select * from T_TravailObjet where [LibelleObjet]
        ➥='bibliothèque'", ConnexionOLE)
        ds = New DataSet()
        da.Fill(ds, "mytable")
        Return ds
    End Function

    <WebMethod()> Public Function AfficheSQL(ByVal st As String) As DataSet
        Connexion = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=
        ➥C:\\bestofgestion97.mdb"
        ConnexionOLE.ConnectionString = Connexion
```

```
            da = New OleDbDataAdapter(st, ConnexionOLE)
            ds = New DataSet()
            da.Fill(ds, "mytable")
            Return ds
        End Function
    End Class
```

Cliquez sur le bouton Démarrer le débogage pour lancer le service web. Une icône dans la barre de notification indique que le service web est actif (voir Figure 12.16) et une page web signale que trois méthodes sont accessibles dans le service (voir Figure 12.17).

Figure 12.16
Dans cet exemple, le service web est accessible sur le port 1202.

Figure 12.17
Les trois méthodes du service web.



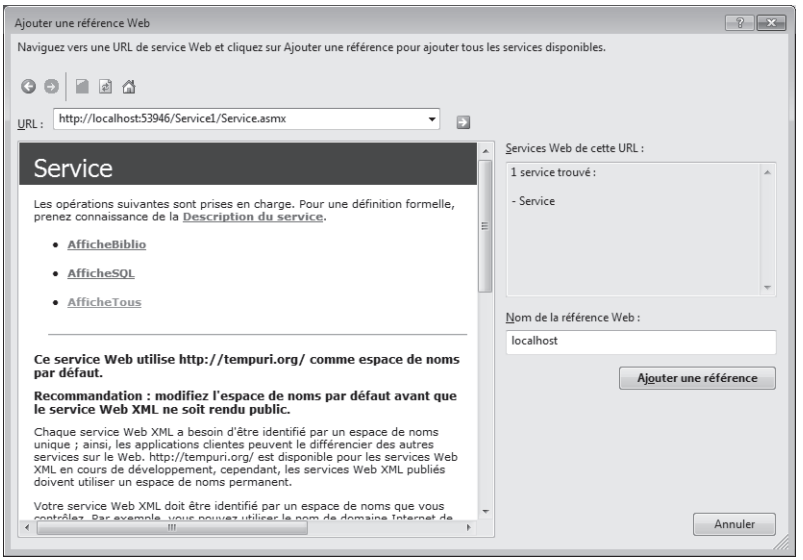
Vous allez maintenant définir une application qui utilise les trois méthodes du service web précédent. Ouvrez une nouvelle instance de Visual Studio et lancez la commande Nouveau/Site web du menu Fichier. Choisissez .NET Framework 3.5 dans la première liste déroulante, sélectionnez le modèle Site Web ASP.NET, donnez le nom TestBDDistante2 à l'application et validez.

Une fois que l'interface de l'application a été créée par Visual Studio.NET, cliquez du bouton droit sur l'entrée C:\…\TestBDDistante2 dans l'Explorateur de solutions et sélectionnez Ajouter une référence web dans le menu surgissant. Entrez l'adresse correspondant

au service web local créé précédemment dans la zone URL de la boîte de dialogue Ajouter une référence web et validez en cliquant sur Aller à.

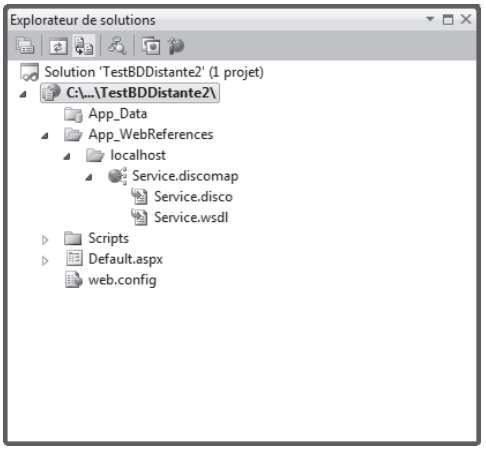
Quelques instants plus tard, les méthodes `AfficheBiblio()`, `AfficheSQL()` et `AfficheTous()` proposées par le service sont référencées dans la boîte de dialogue Ajouter une référence web (voir Figure 12.18).

Figure 12.18
`Service1` *donne accès à trois méthodes.*



Cliquez sur Ajouter une référence. L'Explorateur de solutions laisse apparaître le nouveau service (voir Figure 12.19).

Figure 12.19
Le nouveau service est bien référencé dans l'application.



En utilisant l'onglet Standard de la Boîte à outils, ajoutez trois boutons de commande et une zone de texte à la feuille de l'application. Modifiez les propriétés de ces contrôles comme indiqué dans le tableau suivant :

Contrôle	*Propriété*	*Valeur*
`Button1`	`Text`	Affiche tous
`Button2`	`Text`	Affiche bibliothèques
`Button3`	`Text`	Exécute SQL
`TextBox1`	`Text`	

Sélectionnez l'onglet Données dans la Boîte à outils et déposez un contrôle `GridView` sur la feuille de l'application. Une boîte de dialogue vous invite à définir la source des données à afficher. Appuyez sur la touche Échap du clavier : la définition de ce contrôle se fera dans le code.

La fenêtre de conception doit à présent avoir l'allure de la Figure 12.20.

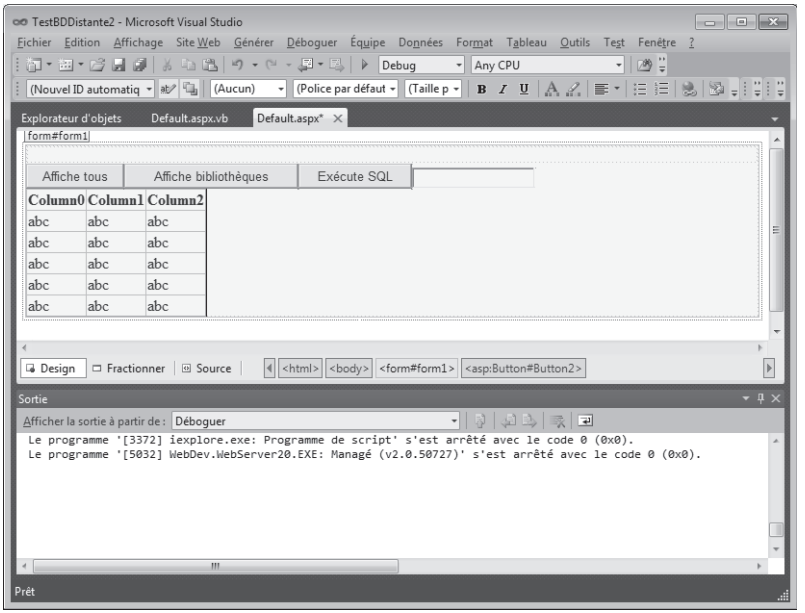


Figure 12.20
La feuille de l'application, en mode Édition.

Double-cliquez sur une partie non occupée de la feuille de l'application et complétez la procédure `Page_Load()` comme suit :

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
➥Handles Me.Load
    Dim objds As New localhost.Service
    Dim ds As DataSet
    ds = objds.AfficheTous
    GridView1.DataSource = ds
    GridView1.DataBind()
End Sub
```

La première instruction donne accès au service web `Service` au travers de l'objet `objds` :

```
Dim objds As New localhost.Service
```

Après avoir défini un objet `DataSet ds`, cet objet est initialisé grâce à la fonction `AfficheTous` du service web :

```
Dim ds As DataSet
ds = objds.AfficheTous
```

Il suffit maintenant de remplir le `GridView` de la feuille avec le `DataSet ds` :

```
GridView1.DataSource = ds.Tables(0).DefaultView
GridView1.DataBind()
```

Pour pouvoir définir l'objet `DataSet ds`, vous devez insérer une instruction `Imports` au début du code :

```
Imports System.Data
```

Il ne reste plus qu'à donner vie aux boutons de commande et à la zone de texte pour terminer l'application. Fermez la fenêtre du navigateur pour revenir à l'environnement de développement.

Double-cliquez sur le premier bouton de commande et complétez la procédure `Button1_Click()` comme suit :

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
➥Handles Button1.Click
    Dim objds As New localhost.Service
    Dim ds As DataSet
    ds = objds.AfficheTous
    GridView1.DataSource = ds
    GridView1.DataBind()
End Sub
```

Les instructions de cette procédure sont les mêmes que celles exécutées au démarrage de l'application. Elles affichent donc la totalité des enregistrements de la table.

Double-cliquez sur le deuxième bouton de commande et complétez la procédure `Button2_Click()` comme suit :

```
Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
➥Handles Button2.Click
    Dim objds As New localhost.Service
    Dim ds As DataSet
    ds = objds.AfficheBiblio
    GridView1.DataSource = ds.Tables(0).DefaultView
    GridView1.DataBind()
End Sub
```

Cette procédure est analogue à `Button1_Click()`, à ceci près que le `DataSet` est initialisé avec la fonction web `AfficheBiblio()` :

```
ds = objds.AfficheBiblio
```

Enfin, double-cliquez sur le troisième bouton de commande et complétez la procédure `Button3_Click()` comme suit :

```
Protected Sub Button3_Click(ByVal sender As Object, ByVal e As System.EventArgs)
➥Handles Button3.Click
    Dim objds As New localhost.Service
    Dim ds As DataSet
    ds = objds.AfficheSQL(TextBox1.Text)
    GridView1.DataSource = ds.Tables(0).DefaultView
    GridView1.DataBind()
End Sub
```

Cette procédure est analogue aux deux précédentes, mais elle remplit le contrôle `DataSet ds` en interrogeant la fonction `AfficheSQL()` (l'argument fourni à la fonction `AfficheSQL()` est le contenu du `TextBox`) :

```
ds = objds.AfficheSQL(TextBox1.Text)
```

Voici le listing complet de l'application. Vous le trouverez, ainsi que les fichiers complémentaires, dans le dossier TestBDDistante2.

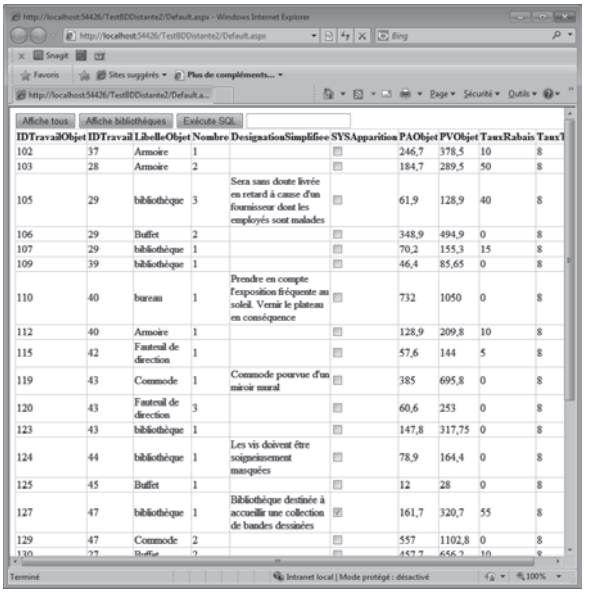
```
Imports System.Data

Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.
    ➥EventArgs) Handles Button1.Click
        Dim objds As New localhost.Service
```

```
        Dim ds As DataSet
        ds = objds.AfficheTous
        GridView1.DataSource = ds
        GridView1.DataBind()
    End Sub

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    ➥Handles Me.Load
        Dim objds As New localhost.Service
        Dim ds As DataSet
        ds = objds.AfficheTous
        GridView1.DataSource = ds
        GridView1.DataBind()
    End Sub

    Protected Sub Button2_Click(ByVal sender As Object, ByVal e As System.
    ➥EventArgs) Handles Button2.Click
        Dim objds As New localhost.Service
        Dim ds As DataSet
        ds = objds.AfficheBiblio
        GridView1.DataSource = ds.Tables(0).DefaultView
        GridView1.DataBind()
    End Sub

    Protected Sub Button3_Click(ByVal sender As Object, ByVal e As System.
    ➥EventArgs) Handles Button3.Click
        Dim objds As New localhost.Service
        Dim ds As DataSet
        ds = objds.AfficheSQL(TextBox1.Text)
        GridView1.DataSource = ds.Tables(0).DefaultView
        GridView1.DataBind()
    End Sub
End Class
```

Appuyez sur la touche F5 pour exécuter l'application. Les Figures 12.21, 12.22 et 12.23 représentent le formulaire web après avoir appuyé (respectivement) sur les boutons Affiche tous, Affiche Bibliothèques et Exécute SQL. Dans le dernier cas, la requête SQL exécutée était la suivante :

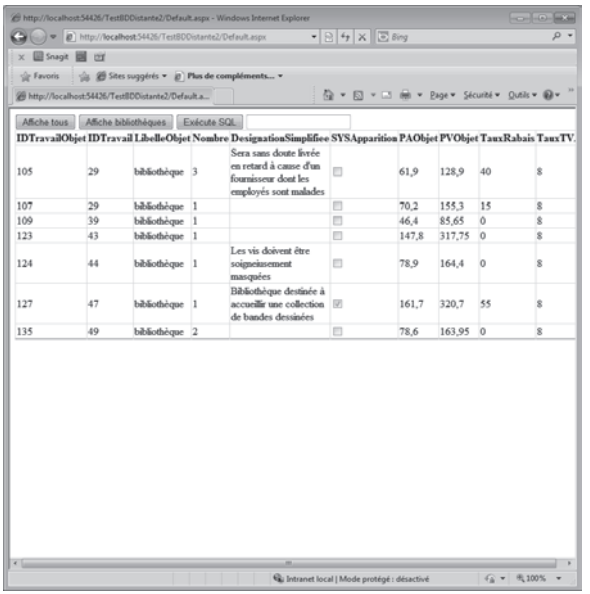
```
select * from T_TravailObjet where [IdTravail]=29
```

Figure 12.21
Tous les enregistrements de la table `Catégories` *s'affichent.*



IDTravailObjet	IDTravail	LibelleObjet	Nombre	DesignationSimplifiee	SYSApparition	PAObjet	PVObjet	TauxRabais	TauxT
102	37	Armoire	1			246,7	378,5	10	8
103	28	Armoire	2			184,7	289,5	50	8
105	29	bibliothèque	3	Sera sans doute livrée en retard à cause d'un fournisseur dont les employés sont malades		61,9	128,9	40	8
106	29	Buffet	2			348,9	494,9	0	8
107	29	bibliothèque	1			70,2	155,3	15	8
109	39	bibliothèque	1			46,4	85,65	0	8
110	40	bureau	1	Prendre en compte l'exposition fréquente au soleil. Vernir le plateau en conséquence		732	1050	0	8
112	40	Armoire	1			128,9	209,8	10	8
115	42	Fauteuil de direction	1			57,6	144	5	8
119	43	Commode	1	Commode pourvue d'un miroir mural		385	695,8	0	8
120	43	Fauteuil de direction	3			60,6	253	0	8
123	43	bibliothèque	1			147,8	317,75	0	8
124	44	bibliothèque	1	Les vis doivent être soigneusement masquées		78,9	164,4	0	8
125	45	Buffet	1			12	28	0	8
127	47	bibliothèque	1	Bibliothèque destinée à accueillir une collection de bandes dessinées		161,7	320,7	55	8
129	47	Commode	2			557	1102,8	0	8

Figure 12.22
Seuls les enregistrements dont le champ `Libelle-Objet` *vaut «*`bibliothèque`*» s'affichent.*



IDTravailObjet	IDTravail	LibelleObjet	Nombre	DesignationSimplifiee	SYSApparition	PAObjet	PVObjet	TauxRabais	TauxTV
105	29	bibliothèque	3	Sera sans doute livrée en retard à cause d'un fournisseur dont les employés sont malades		61,9	128,9	40	8
107	29	bibliothèque	1			70,2	155,3	15	8
109	39	bibliothèque	1			46,4	85,65	0	8
123	43	bibliothèque	1			147,8	317,75	0	8
124	44	bibliothèque	1	Les vis doivent être soigneusement masquées		78,9	164,4	0	8
127	47	bibliothèque	1	Bibliothèque destinée à accueillir une collection de bandes dessinées		161,7	320,7	55	8
135	49	bibliothèque	2			78,6	163,95	0	8

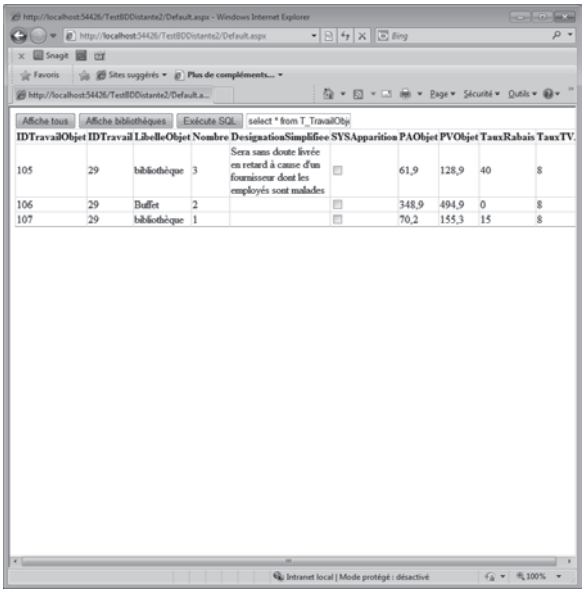


Figure 12.23
Seuls les enregistrements correspondant à la requête SQL apparaissent.

Accès à une base de données *via* LINQ

Les bases de données ADO.NET, SQL Server et SQL Server Express peuvent être accédées par l'intermédiaire de requêtes LINQ.

Voici, à titre d'exemple une requête permettant de sélectionner des enregistrements dans une table d'une base de données ADO.Net :

```
Dim objets As DataTable = ds.Tables("T_TravailObjet")
Dim query = _
  From objet In objets.AsEnumerable() _
  Where objet.Field(Of String)("LibelleObjet") = "bibliothèque" _
  Select New With { _
      .IDTravailObjet = objet.Field(Of Integer)("IDTravailObjet"), _
      .IDTravail = objet.Field(Of Integer)("IDTravail"), _
      .Nombre = objet.Field(Of Integer)("Nombre")
    }
```

Cette requête crée un iEnumerable contenant les données `IDTravailObjet`, `IDTravail` et `Nombre`. Ces données proviennent des champs `IDTravailObjet`, `IDTravail` et `Nombre` de la table `T_TravailObjet`. Seuls les enregistrements dont le champ `LibelleObjet` vaut "Bibliothèque" sont sélectionnés par la requête.

Vous vous reporterez à la section intitulée "LINQ to DataSet", au Chapitre 27 pour avoir des informations complémentaire sur l'interrogation de bases de données ADO.NET.

Dans cet autre exemple, la table interrogée est de type SQL Server :

```
Dim query = From orde In db.Orders
            Order By orde.CustomerID
            Select orde.CustomerID, orde.OrderID
```

La requête LINQ extrait les champs `CustomerID` et `OrderID` de la table `Orders` et les classe par ordre croissant sur le champ `CustomerID`.

Pour avoir des informations complémentaires concernant le requêtage LINQ des bases de données SQL Server et SQL Server Express, vous consulterez la section intitulée "LINQ to SQL" au Chapitre 27.

13

Traitements multitâches/ multicœurs

Il y a quelques années, on mesurait la puissance de calcul brut d'un ordinateur à la fréquence de son microprocesseur. Aujourd'hui, cette fréquence se situe la plupart du temps autour des 3 GHz. En l'état actuel de la technologie, cette barrière a du mal à être franchie, essentiellement pour des raisons de dissipation thermique. Pour augmenter la puissance de calcul, les fondeurs ont eu l'idée de loger plusieurs puces (ou cœurs) dans un même chip. Cette solution, fort prometteuse, n'est encore qu'assez mal utilisée par les éditeurs de logiciels qui, bien souvent, se contentent d'utiliser des instructions traditionnelles, de type monocœur. Heureusement, avec Visual Basic 2010, la mise en place d'un traitement multitâches/ multicœurs est un vrai jeu d'enfant. Cette facilité de mise en œuvre découle de plusieurs techniques :

- Le composant BackgroundWorker : grâce à lui, un traitement consommateur de temps machine peut être implémenté dans un thread qui s'exécutera en tâche de fond. Cela évite de "geler" la machine si un long calcul doit être effectué, et le programme donne une impression de fluidité à ses utilisateurs.

- Le .NET Framework 4 fournit de nouvelles bibliothèques de classes. En particulier, la classe `Parallel` qui permet de répartir l'exécution de boucles `For` et `For Each` sur les différents cœurs disponibles.
- Le concept de tâches, lié à la classe `Task`, permet de gérer finement l'exécution de code par les différents cœurs disponibles.

Deux applications illustrent le composant BackgroundWorker sont développées dans ce chapitre. La première montre comment effectuer un calcul "lourd" en tâche de fond. La seconde propose deux techniques pour exécuter une application externe : en tâche de fond ou en tâche principale.

Deux applications illustrent l'utilisation de l'instruction `Parallel.For` et montrent sa supériorité par rapport à une classique structure `For Next`.

Enfin, une application montre comment affecter l'exécution de blocs de code à certains cœurs du processeur utilisé. Les instructions de cette application n'ont aucune incidence sur un processeur doté d'un seul cœur.

Exécution d'un calcul en tâche de fond

Cette première application va calculer une factorielle en tâche de fond. Lancez commande Nouveau/Projet du menu Fichier. Choisissez .NET Framework 4 dans la liste déroulante Framework. Sélectionnez Visual Basic/Windows dans le volet gauche et Application Windows Forms dans le volet central. Donnez le nom Multitâche au projet et cliquez sur OK. Ajoutez deux `Label`, un `TextBox`, deux `Button`, un `ProgressBar` et un `BackgroundWorker` (ce contrôle se trouve sous l'onglet `Composants` de la Boîte à outils) pour obtenir l'effet visuel de la Figure 13.1.

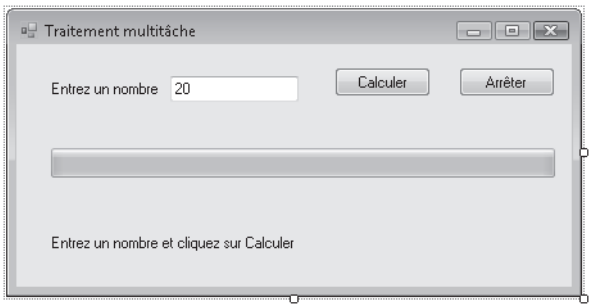


Figure 13.1
Agencement de l'application de traitement multitâche.

Modifiez les propriétés de ces contrôles comme indiqué dans le tableau suivant :

Contrôle	*Propriété*	*Valeur*
Form1	Text	Traitement multitâche
Label1	Text	Entrez un nombre
TextBox1	Text	20
Button1	Text	Calculer
Button2	Text	Arrêter
Label2	Text	Entrez un nombre et cliquez sur Calculer
BackgroundWorker1	Name	bgw
BackgroundWorker1	WorkerReportsProgress	True

Double-cliquez sur le bouton Calculer et complétez la procédure `Button1_Click()` comme suit :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Button1.Click
    If CInt(TextBox1.Text) > 20 Or CInt(TextBox1.Text) < 1 Then
        MsgBox("Le nombre doit être compris entre 1 et 20")
    Else
        Label2.Text = "Calcul en cours..."
        Button1.Enabled = False
        bgw.RunWorkerAsync(CInt(TextBox1.Text))
    End If
End Sub
```

La première instruction teste la valeur entrée par l'utilisateur dans la zone de texte. Pour éviter un dépassement dans le calcul de la factorielle, cette valeur doit être comprise entre 1 et 20. Dans le cas contraire, un message d'erreur apparaît :

```
If CInt(TextBox1.Text) > 20 Or CInt(TextBox1.Text) < 1 Then
    MsgBox("Le nombre doit être compris entre 1 et 20")
```

Si la valeur entrée est comprise entre 1 et 20, trois actions sont effectuées. Dans un premier temps, un message s'affiche dans la fenêtre par l'intermédiaire du contrôle `Label2` :

```
Else
    Label2.Text = "Calcul en cours..."
```

Dans un deuxième temps, le bouton Calculer est désactivé pour éviter que l'utilisateur n'appuie dessus avant la fin du calcul :

```
Button1.Enabled = False
```

Enfin, dans un troisième temps, le processus asynchrone démarre :

```
bgw.RunWorkerAsync(CInt(TextBox1.Text))
```

Cette dernière instruction génère un événement `Dowork` rattaché à l'objet `BackgroundWorker bgw`. Nous allons donc définir la procédure événementielle `bgw_Dowork()`. Cliquez sur l'icône bgw (en dessous de la fenêtre de l'application), affichez les événements dans la fenêtre des propriétés et double-cliquez sur la propriété `DoWork` pour accéder à la procédure `bgw_DoWork()`. Complétez cette procédure comme suit :

```
Private Sub bgw_DoWork(ByVal sender As System.Object, ByVal e As System.
➥ComponentModel.DoWorkEventArgs) Handles bgw.DoWork
    Dim worker As BackgroundWorker = CType(sender, BackgroundWorker)
    e.Result = factorielle(e.Argument, worker, e)
End Sub
```

La première instruction récupère l'objet `BackgroundWorker` à l'origine de cet événement :

```
Dim worker As BackgroundWorker = CType(sender, BackgroundWorker)
```

La seconde instruction appelle la fonction `Factorielle()` et stocke la valeur renvoyée dans la propriété `Result` de l'objet `DoWorkEventArgs` :

```
e.Result = factorielle(e.Argument, worker, e)
```

L'étape suivante consiste à définir la fonction `Factorielle()` :

```
Function factorielle(ByVal n As Integer, ByVal worker As BackgroundWorker,
➥ByVal e As DoWorkEventArgs) As Long
    Dim i As Long
    Dim percentcomplete As Integer
    resultat = 1
    For i = 1 To n
        resultat = i * resultat
        percentcomplete = CSng(i) / CSng(n) * 100
        worker.ReportProgress(percentcomplete)
        System.Threading.Thread.Sleep(150)
    Next (i)
    Return resultat
End Function
```

Après avoir déclaré l'index de la boucle de calcul, la variable représentant l'état d'avancement dans le calcul et la variable dans laquelle le résultat va être stocké :

```
Dim i As Long
Dim percentcomplete As Integer
resultat = 1
```

le calcul récursif de la factorielle prend place :

```
For i = 1 To n
    resultat = i * resultat
    percentcomplete = CSng(i) / CSng(n) * 100
    worker.ReportProgress(percentcomplete)
    System.Threading.Thread.Sleep(150)
Next (i)
```

Remarquez les deux dernières instructions de la boucle :

- `worker.ReportProgress(percentcomplete)` affiche l'état d'avancement du calcul. Cette instruction va déclencher l'événement `ProgressChanged` de l'objet `BackgroundWorker`.
- `System.Threading.Thread.Sleep(150)` introduit une pause de 150 millisecondes entre chacune des étapes du calcul. Cela évite un avancement trop rapide de la barre de progression.

L'étape suivante consiste à définir la procédure événementielle `bgw_ProgressChanged()`. Cliquez sur l'icône bgw (en dessous de la fenêtre de l'application), affichez les événements dans la fenêtre des propriétés et double-cliquez sur la propriété `ProgressChanged` pour accéder à la procédure `bgw_ProgressChanged()`. Complétez cette procédure comme suit :

```
Private Sub bgw_ProgressChanged(ByVal sender As System.Object, ByVal e As
➥System.ComponentModel.ProgressChangedEventArgs) Handles bgw.ProgressChanged
    Me.ProgressBar1.Value = e.    Private Sub bgw_ProgressChanged(ByVal sender
    ➥As System.Object, ByVal e As System.ComponentModel.ProgressChangedEventArgs)
    ➥Handles bgw.ProgressChanged
        Me.ProgressBar1.Value = e.ProgressPercentage
    End Sub
End Sub
```

L'unique instruction de cette procédure met à jour la barre de progression en utilisant la variable `ProgressPercentage`.

Nous allons maintenant définir la procédure à exécuter en fin de calcul. Cliquez sur l'icône bgw (en dessous de la fenêtre de l'application), affichez les événements dans la fenêtre des propriétés et double-cliquez sur la propriété `RunWorkerCompleted` pour accéder à la procédure `bgw_ RunWorkerCompleted()`. Complétez cette procédure comme suit :

```
Private Sub bgw_RunWorkerCompleted(ByVal sender As System.Object, ByVal e As
➥System.ComponentModel.RunWorkerCompletedEventArgs) Handles bgw.RunWorkerCompleted
    Button1.Enabled = True
    Label2.Text = "Factorielle(" + TextBox1.Text + ") = " + Str(resultat)
End Sub
```

Cette procédure active à nouveau le bouton Calculer et affiche le résultat du calcul dans le contrôle `Label2`.

Pour terminer, double-cliquez sur le bouton Arrêter et ajoutez une instruction `End` dans la procédure `Button2_Click` :

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Button2.Click
    End
End Sub
```

L'application est maintenant totalement opérationnelle. Vous pouvez l'exécuter en appuyant sur la touche F5. Entrez un nombre compris entre 1 et 20 dans la zone de texte et cliquez sur le bouton Calculer. Vous pouvez voir l'avancement de la barre de progression – en ayant bien en tête que les calculs sont effectués dans un thread autre que celui de l'application (voir Figure 13.2).

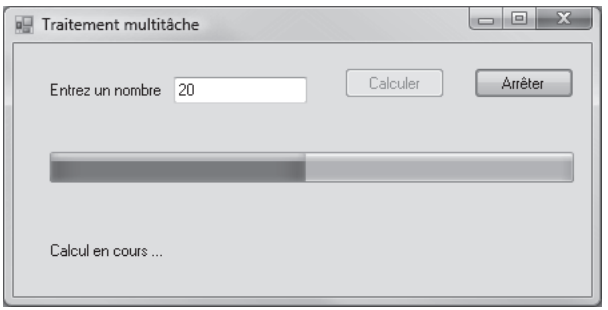


Figure 13.2
L'application Traitement multitâche en cours d'exécution.

Voici le code complet de l'application. Vous trouverez les fichiers correspondants dans le dossier multitâche.

```
Imports System.ComponentModel
Public Class Form1
    Dim longueur As Integer = 0
    Dim resultat As Long = 0

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Button1.Click
        If CInt(TextBox1.Text) > 20 Or CInt(TextBox1.Text) < 1 Then
            MsgBox("Le nombre doit être compris entre 1 et 20")
        Else
            Label2.Text = "Calcul en cours..."
            Button1.Enabled = False
```

```
            bgw.RunWorkerAsync(CInt(TextBox1.Text))
        End If
    End Sub
    Private Sub bgw_DoWork(ByVal sender As System.Object, ByVal e As System.
    ➥ComponentModel.DoWorkEventArgs) Handles bgw.DoWork
        Dim worker As BackgroundWorker = CType(sender, BackgroundWorker)
        e.Result = factorielle(e.Argument, worker, e)
    End Sub
    Function factorielle(ByVal n As Integer, ByVal worker As BackgroundWorker,
    ➥ByVal e As DoWorkEventArgs) As Long
        Dim i As Long
        Dim percentcomplete As Integer
        resultat = 1
        For i = 1 To n
            resultat = i * resultat
            percentcomplete = CSng(i) / CSng(n) * 100
            worker.ReportProgress(percentcomplete)
            System.Threading.Thread.Sleep(150)
        Next (i)
        Return resultat
    End Function

    Private Sub bgw_ProgressChanged(ByVal sender As System.Object, ByVal e As
    ➥System.ComponentModel.ProgressChangedEventArgs) Handles bgw.ProgressChanged
        Me.ProgressBar1.Value = e.ProgressPercentage
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Button2.Click
        End
    End Sub

    Private Sub bgw_RunWorkerCompleted(ByVal sender As System.Object, ByVal
    ➥e As System.ComponentModel.RunWorkerCompletedEventArgs) Handles bgw.
    ➥RunWorkerCompleted
        Button1.Enabled = True
        Label2.Text = "Factorielle(" + TextBox1.Text + ") = " + Str(resultat)
    End Sub

End Class
```

Exécution d'une application externe

Il est parfois utile d'appeler une application exécutable dans un projet Visual Basic. La technique à utiliser diffère selon que vous vouliez lancer l'application en tant que processus ou attendre sa fermeture pour reprendre le contrôle.

Définissez un nouveau projet fondé sur le modèle Application Windows Forms et donnez-lui le nom LanceBN. Insérez un `TextBox` et deux `Button` sur la feuille du projet. Modifiez les propriétés de ces contrôles comme indiqué dans le tableau suivant :

Contrôle	*Propriété*	*Valeur*
`Form1`	`Text`	Lanceur d'applications
`TextBox1`	`Text`	Notepad.exe
`Button1`	`Text`	Lancer
`Button2`	`Text`	Lancer et attendre

Modifiez l'emplacement et les dimensions de ces contrôles de sorte que la feuille de l'application ressemble à la Figure 13.3.

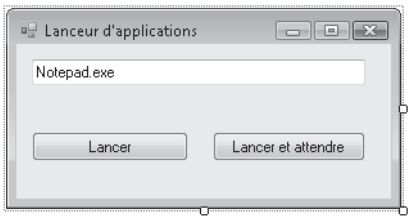


Figure 13.3
La feuille de l'application, en mode Édition.

Double-cliquez sur le bouton Lancer et complétez la procédure `Button1_Click()` comme suit :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Button1.Click
    Process.Start(TextBox1.Text)
    MessageBox.Show("L'application a été lancée", "Information")
End Sub
```

La classe `Process` permet de démarrer des applications en tant que processus, donc sans bloquer le programme qui les active. Consultez la documentation en ligne pour connaître les nombreuses méthodes de cette classe.

La première instruction lance le processus dont le nom est spécifié dans la zone de texte `TextBox1` :

```
Process.Start(TextBox1.Text)
```

La seconde instruction affiche une boîte de message indiquant que l'application a été lancée. Étant donné que le lancement d'un processus ne bloque pas l'ordinateur, cette boîte de message apparaît immédiatement :

```
MessageBox.Show("L'application a été lancée", "Information")
```

Double-cliquez sur le bouton Lancer et attendre, et complétez la procédure `Button2_Click()` comme suit :

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
➥Handles Button2.Click
    Shell(TextBox1.Text, AppWinStyle.NormalFocus, True)
    MessageBox.Show("L'application vient d'être fermée", "Information")
End Sub
```

La première instruction utilise la procédure `Shell()` pour lancer l'application spécifiée dans la zone de texte :

```
Shell(TextBox1.Text, AppWinStyle.NormalFocus, True)
```

Le premier argument de la procédure précise le nom de l'application à lancer. Ce nom doit être un chemin complet si l'application ne se trouve pas dans le dossier d'installation de Windows ou dans le dossier système.

Le deuxième argument indique l'état de l'application. Il peut être égal à l'une des valeurs suivantes :

- `AppWinStyle.Hide`. Fenêtre cachée ayant le focus.
- `AppWinStyle.NormalFocus`. Fenêtre normale ayant le focus.
- `AppWinStyle.MinimizedFocus`. Fenêtre minimisée dans la barre des tâches ayant le focus.
- `AppWinStyle.MaximizedFocus`. Fenêtre ouverte et maximisée.
- `AppWinStyle.NormalNoFocus`. Fenêtre normale n'ayant pas le focus.
- `AppWinStyle.MinimizedNoFocus`. Fenêtre minimisée dans la barre des tâches n'ayant pas le focus.

Enfin, le troisième argument indique si le programme parent Visual Basic doit attendre (`True`) ou ne pas attendre (`False`) la fin de l'application lancée pour poursuivre son exécution. Si ce paramètre n'est pas spécifié, la fin de l'application n'est pas attendue.

La deuxième instruction affiche un message qui précise que l'application vient d'être fermée. Ce message ne s'affiche qu'après la fermeture de l'application, car le troisième paramètre de la fonction `Shell()` vaut `True` :

```
MessageBox.Show("L'application vient d'être fermée", "Information")
```

Lorsque l'instruction `Shell` ne contient que deux paramètres, l'application est lancée et le contrôle est immédiatement redonné à l'instruction suivante. Ce comportement fait ressembler la procédure `Shell()` au démarrage d'un processus. Cependant, les possibilités des processus sont bien plus nombreuses. Pour en savoir plus à ce sujet, consultez la documentation de Visual Basic.

Voici le listing complet de l'application. Le projet se trouve dans le dossier LanceBN après installation des sources de l'ouvrage.

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Button1.Click
        Process.Start(TextBox1.Text)
        MessageBox.Show("L'application a été lancée", "Information")
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.
    ➥EventArgs) Handles Button2.Click
        Shell(TextBox1.Text, AppWinStyle.NormalFocus, True)
        MessageBox.Show("L'application vient d'être fermée", "Information")
    End Sub
End
```

Parallélisation des boucles

La classe `Parallel` du .NET Framework 4 permet de répartir l'exécution de boucles `For` et `For Each` sur les différents cœurs disponibles. Pour illustrer cette classe, nous allons développer une application console qui calcule un million de racines carrées, dans un premier temps à l'aide d'une boucle `For Next` conventionnelle, puis à l'aide d'une boucle `Parallel.For`. Les temps de calcul s'afficheront à la suite des deux variantes du code.

Lancez la commande Nouveau/Projet dans le menu Fichier. Sélectionnez .NET Framework 4 dans la liste déroulante Framework. Sélectionnez Visual Basic/Windows dans le volet

gauche. Cliquez sur Application console, donnez le nom ParallelFor1 à l'application et validez en cliquant sur OK. Complétez le code avec les instructions suivantes (cette application se trouve dans le dossier Projects\ParallelFor1 des sources de l'ouvrage après leur installation).

```
Imports System.Threading
Imports System.Threading.Tasks
Module Module1

  Function Calcul(ByVal i As Integer, ByVal result As String) As Boolean
    result = "Racine carrée : " + Math.Sqrt(i).ToString()
    Return True
  End Function

  Sub Main()
    Dim result As String
    Dim debut As TimeSpan

    Console.WriteLine("Début du calcul simple thread")
    debut = DateTime.Now.TimeOfDay
    For i = 0 To 10000000
      Calcul(i,result)
    Next
    Console.WriteLine("Fin du calcul simple thread")
    Console.WriteLine(DateTime.Now.TimeOfDay - debut)
    Console.WriteLine()

    Console.WriteLine("Début du calcul multithread")
    debut = DateTime.Now.TimeOfDay
    Parallel.For(0, 10000000, Function(i) Calcul(i, result))
    Console.WriteLine("Fin du calcul multithread")
    Console.WriteLine(DateTime.Now.TimeOfDay - debut)

    Console.ReadKey()
  End Sub

End Module
```

Examinons le code.

Après avoir déclaré les deux espaces de noms nécessaires à l'exécution multithreads :

```
Imports System.Threading
Imports System.Threading.Tasks
```

Le bloc de code suivant définit la fonction `Calcul()`, responsable du calcul des 1 million de racines carrées :

```
Function Calcul(ByVal i As Integer, ByVal result As String) As Boolean
  result = "Racine carrée : " + Math.Sqrt(i).ToString()
  Return True
End Function
```

La procédure `Main()` se décompose en quatre blocs.

Le premier bloc définit les variables utilisées dans la procédure :

```
Dim result As String
Dim debut As TimeSpan
```

Le deuxième bloc effectue une boucle `For Next` traditionnelle qui appelle la fonction `Calcul()` 1 million de fois. Après avoir affiché un message pour indiquer le début du calcul :

```
Console.WriteLine("Début du calcul simple thread")
```

L'heure système est mémorisée dans la variable `debut` :

```
debut = DateTime.Now.TimeOfDay
```

Puis le calcul est lancé :

```
For i = 0 To 10000000
  Calcul(i,result)
Next
```

Lorsque la boucle `For Next` est terminée, une instruction `WriteLine` indique la fin du calcul :

```
Console.WriteLine("Fin du calcul simple thread")
```

Et une autre affiche le temps nécessaire pour effectuer le calcul :

```
Console.WriteLine(DateTime.Now.TimeOfDay - debut)
```

Le troisième bloc d'instructions effectue le même calcul en mode multithreads. Après avoir indiqué le début du calcul et mémorisé l'heure système :

```
Console.WriteLine("Début du calcul multithread")
debut = DateTime.Now.TimeOfDay
```

La fonction `Calcul()` est appelée un million de fois :

```
Parallel.For(0, 10000000, Function(i) Calcul(i, result))
```

Remarquez la syntaxe particulière de cette instruction. Les deux premiers paramètres font référence aux valeurs minimale et maximale de l'index. Le troisième paramètre spécifie la fonction à exécuter et les paramètres qui doivent lui être passés.

Lorsque l'exécution de l'instruction `Parallel.For` est terminée, un message apparaît dans la console, accompagné du temps de calcul :

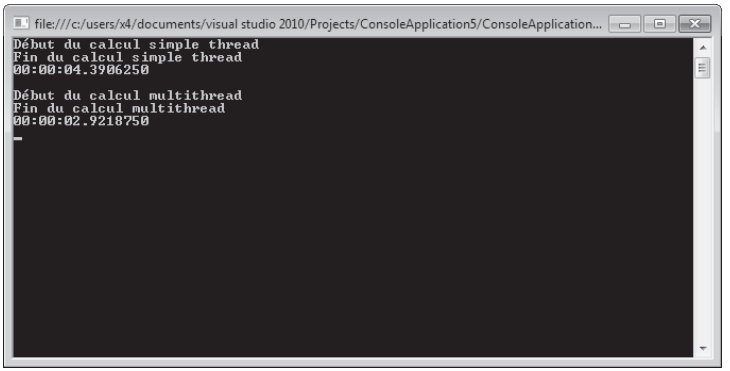
```
Console.WriteLine("Fin du calcul multithread")
Console.WriteLine(DateTime.Now.TimeOfDay - debut)
```

La dernière instruction attend que l'utilisateur appuie sur une touche du clavier pour mettre fin au programme :

```
Console.ReadKey()
```

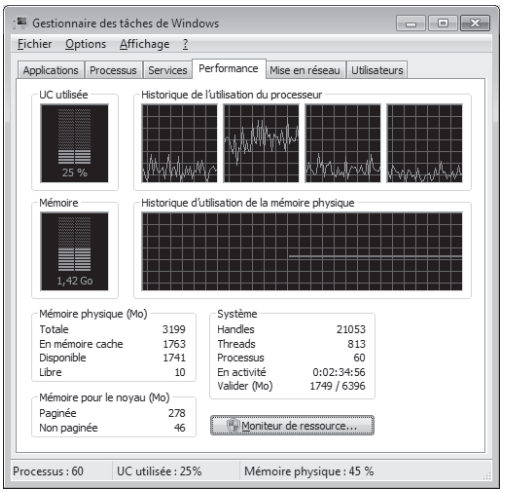
La Figure 13.4 représente le résultat de ce programme exécuté sur un AMD Phenom 9850 cadencé à 2,5 GHz, doté de 3 Go de RAM DDR2 et fonctionnant sous Windows 7 32 bits.

Figure 13.4
Le traitement parallèle est effectivement plus rapide sur un multicœur.



Les Figures 13.5 et 13.6 représentent les deux phases d'exécution du programme.

Figure 13.5
Le deuxième cœur est utilisé pour effectuer le calcul.



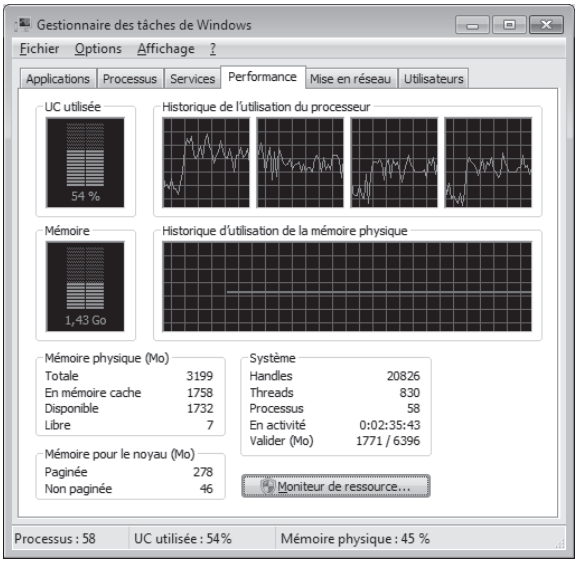


Figure 13.6
Les quatre cœurs effectuent le calcul, améliorant ainsi les performances.

Voici un deuxième exemple qui met en évidence la supériorité de la programmation parallèle lorsqu'il est exécuté sur un ordinateur muni de plusieurs cœurs. Ce code est issu de la librairie en ligne MSDN, page **http://msdn.microsoft.com/en-us/library/dd460713.aspx**. Il consiste en la multiplication de matrices de 180 colonnes et 2 000 lignes. Vous trouverez plusieurs autres exemples de programmation parallèle sur cette page. N'hésitez pas à vous y reporter.

Lancez la commande Nouveau/Projet dans le menu Fichier. Sélectionnez .NET Framework 4 dans la liste déroulante Framework, cliquez sur Application console, donnez le nom ParallelFor2 à l'application et validez en cliquant sur OK. Complétez le code avec les instructions suivantes (cette application se trouve dans le dossier Projects\ParallelFor2 des sources de l'ouvrage après leur installation) :

```
Imports System.Threading.Tasks
Module Module1

  Sub MultiplyMatricesSequential(ByVal matA As Double(,), ByVal matB As Double(,),
  ➥ByVal result As Double(,))
    Dim matACols As Integer = matA.GetLength(1)
    Dim matBCols As Integer = matB.GetLength(1)
    Dim matARows As Integer = matA.GetLength(0)

    For i As Integer = 0 To matARows - 1
      For j As Integer = 0 To matBCols - 1
```