eBook Gratuit

APPRENEZ

Visual Basic .NET Language

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#vb.net

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: visual-basic--net-language

It is an unofficial and free Visual Basic .NET Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Visual Basic .NET Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Premiers pas avec le langage Visual Basic .NET

Remarques

Visual Basic .NET est le successeur officiel du langage de programmation Visual Basic d'origine de Microsoft. Visual Basic [.NET] semble avoir des similitudes avec Python avec l'absence de points-virgules et de crochets, mais partage avec C ++ la structure de base des fonctions. Les accolades sont absentes dans VB .NET, mais remplacées par des phrases comme `End If` , `Next` et `End Sub` .

Versions

Version VB.NET	Version de Visual Studio	Version du framework .NET	Date de sortie
7.0	2002	1.0	2002-02-13
7.1	2003	1.1	2003-04-24
8.0	2005	2,0 / 3,0	2005-10-18
9.0	2008	3.5	2007-11-19
10.0	2010	4.0	2010-04-12
11.0	2012	4.5	2012-08-15
12,0	2013	4.5.1 / 4.5.2	2013-10-17
14.0	2015	4.6.0 ~ 4.6.2	2015-07-20
15.0	2017	4.7	2017-03-07

Examples

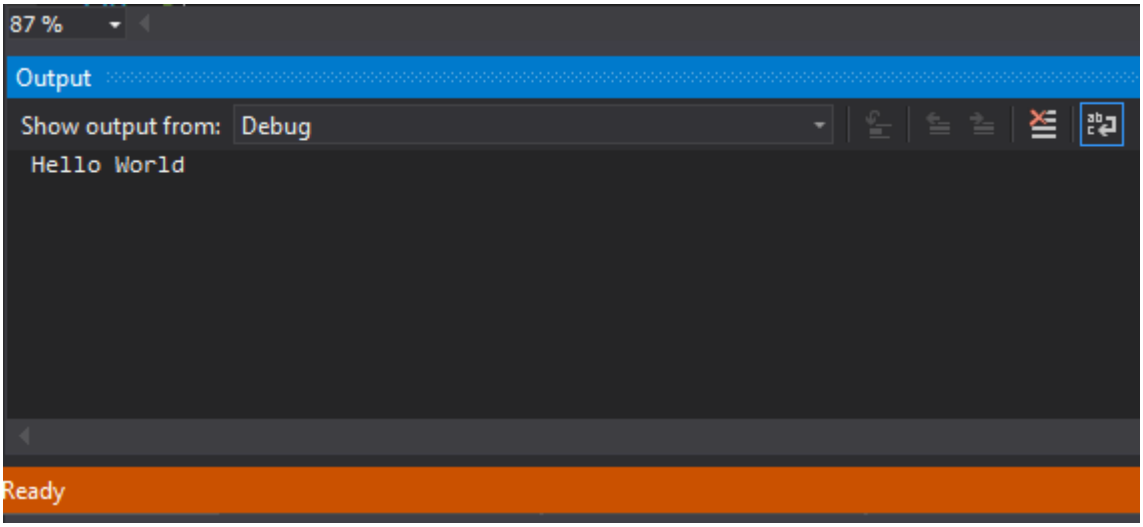
Bonjour le monde

Tout d'abord, installez une version de Microsoft Visual Studio , y compris l'édition communautaire gratuite. Ensuite, créez un projet Application console Visual Basic de type *Application console* et le code suivant imprimera la chaîne `'Hello World'` sur la console:

```
Module Module1
```

```
    Sub Main()
        Console.WriteLine("Hello World")
    End Sub

End Module
```

Ensuite, enregistrez et appuyez sur `F5` sur le clavier (ou accédez au menu *Déboguer* , puis cliquez sur *Exécuter sans débogage* ou *Exécuter*) pour compiler et exécuter le programme. `'Hello World'` devrait apparaître dans la fenêtre de la console.



Bonjour tout le monde sur une zone de texte en cliquant sur un bouton

Faites glisser 1 zone de texte et 1 bouton

Double-cliquez sur le bouton1 et vous serez transféré à l' `Button1_Click event`

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

Tapez le nom de l'objet que vous souhaitez cibler, dans notre cas c'est le `textbox1` . `.Text` est la propriété que nous voulons utiliser si nous voulons y placer un texte.

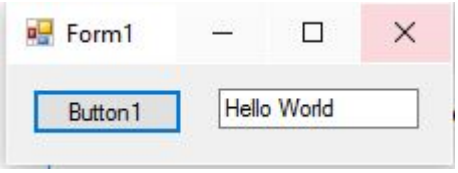
`Property Textbox.Text, gets or sets the current text in the TextBox` . Maintenant, nous avons `Textbox1.Text`

Nous devons définir la valeur de ce `Textbox1.Text` pour que nous `Textbox1.Text` le signe `=` . La valeur que nous voulons mettre dans le `Textbox1.Text` est `Hello World` . Dans l'ensemble, il s'agit du code total pour mettre une valeur de `Hello World` dans `Textbox1.Text`

```
TextBox1.Text = "Hello World"
```

Ajout de ce code à l' `clicked event` de `button1`

```
Public Class Form1
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        TextBox1.Text = "Hello World"
    End Sub
End Class
```

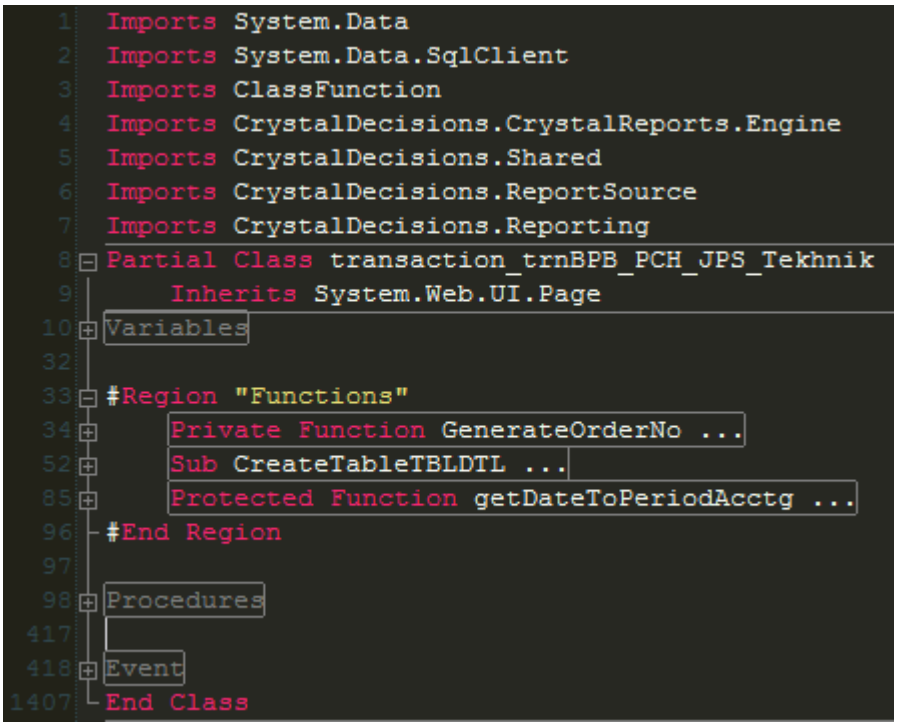


Région

Par souci de lisibilité, ce qui sera utile pour les débutants lors de la lecture du code VB aussi bien pour les développeurs à temps plein que pour maintenir le code, nous pouvons utiliser "Region" pour définir une région du même ensemble d'événements, fonctions ou variables:

```
#Region "Events"
    Protected Sub txtPrice_TextChanged(...) Handles txtPrice.TextChanged
        'Do the ops here...
    End Sub

    Protected Sub txtTotal_TextChanged(...) Handles txtTotal.TextChanged
        'Do the ops here...
    End Sub

    'Some other events....

#End Region
```

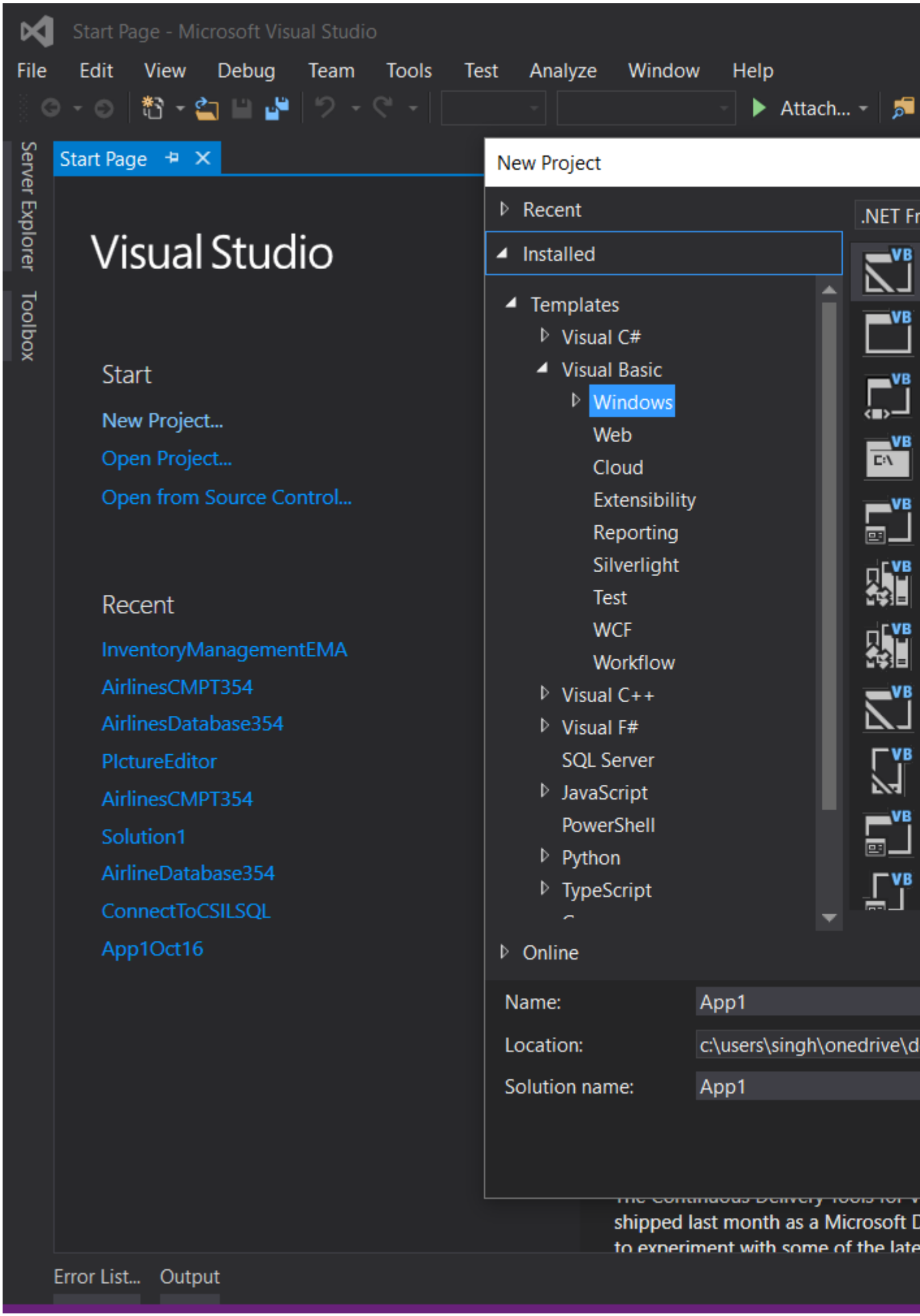
Ce bloc de région peut être réduit pour obtenir une aide visuelle lorsque la ligne de code atteint 1000+. Il est également enregistrer vos efforts de défilement.



Testé sur VS 2005, 2008 2010, 2015 et 2017.

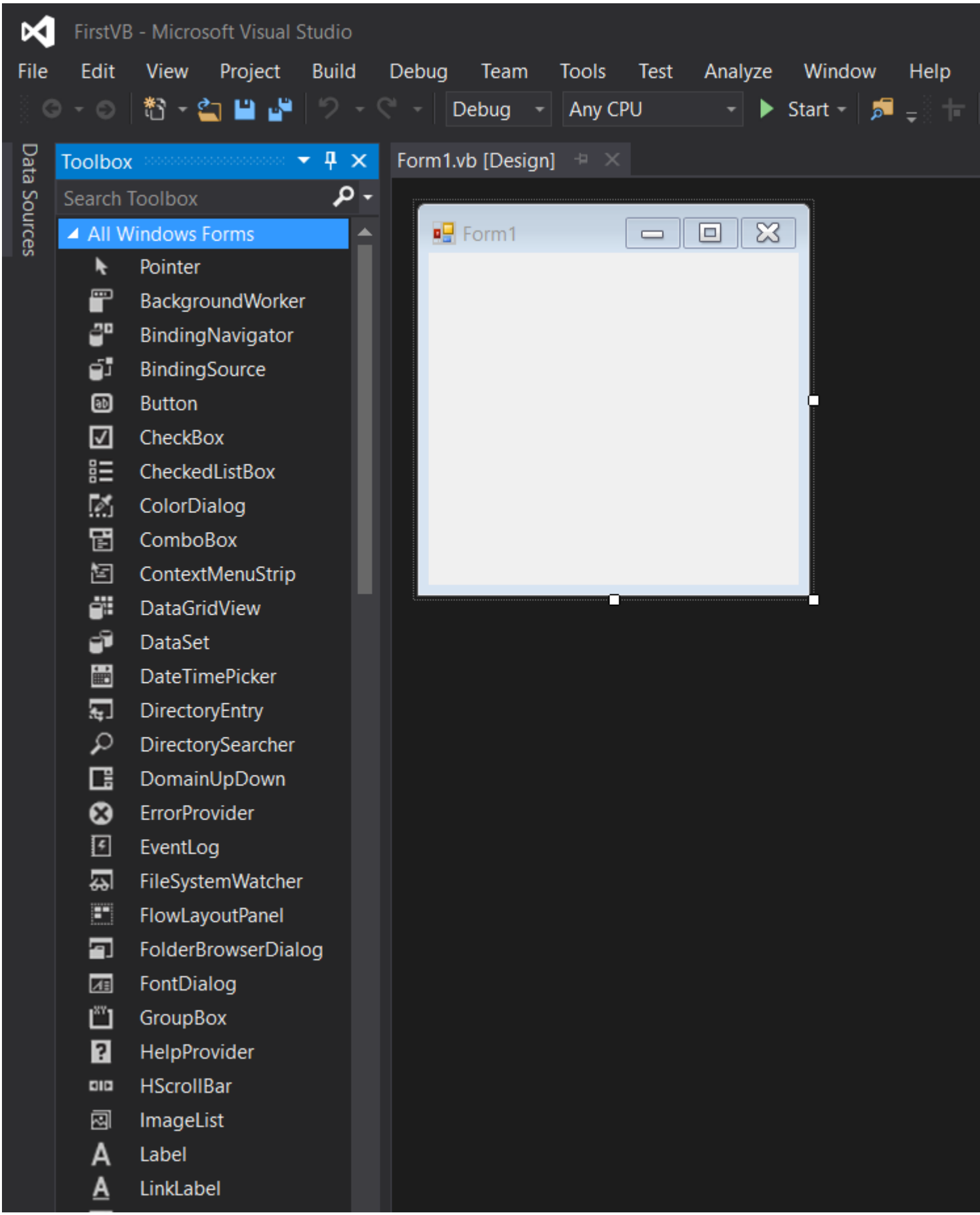
Créer un simple calculateur pour vous familiariser avec l'interface et le code.

1. Une fois que vous avez installé Visual Studio à partir de https://www.visualstudio.com/downloads/ , démarrez un nouveau projet.

2. 
Start Page - Microsoft Visual Studio
File Edit View Debug Team Tools Test Analyze Window Help
Attach...
Server Explorer
Toolbox
Start Page
Visual Studio
Start
New Project...
Open Project...
Open from Source Control...
Recent
InventoryManagementEMA
AirlinesCMPT354
AirlinesDatabase354
PIctureEditor
AirlinesCMPT354
Solution1
AirlineDatabase354
ConnectToCSILSQL
App1Oct16
New Project
Recent
Installed
Templates
Visual C#
Visual Basic
Windows
Web
Cloud
Extensibility
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
SQL Server
JavaScript
PowerShell
Python
TypeScript
Online
Name: App1
Location: c:\users\singh\onedrive\d
Solution name: App1
shipped last month as a Microsoft D
Error List... Output

3. Sélectionnez «Application Windows Forms» dans Visual Basic Tab. Vous pouvez le renommer ici si vous en avez besoin.

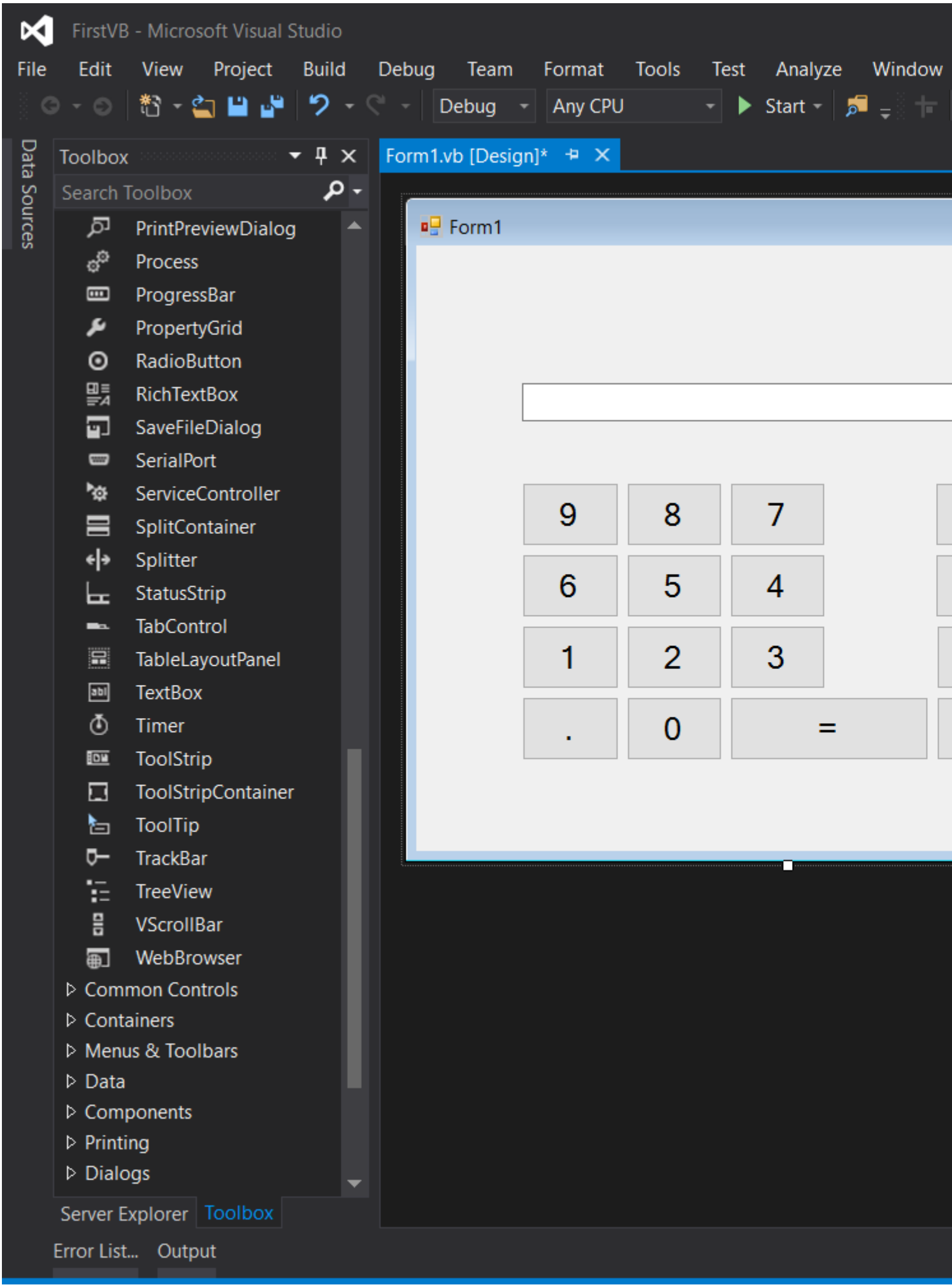
4. Une fois que vous cliquez sur "OK", vous verrez cette fenêtre:



5. Cliquez sur l'onglet "Boîte à outils" à gauche. La barre d'outils a l'option "masquer automatiquement" activée par défaut. Pour désactiver cette option, cliquez sur le petit

symbole situé entre le symbole «flèche vers le bas» et le symbole «x», dans le coin supérieur droit de la fenêtre Boîte à outils.

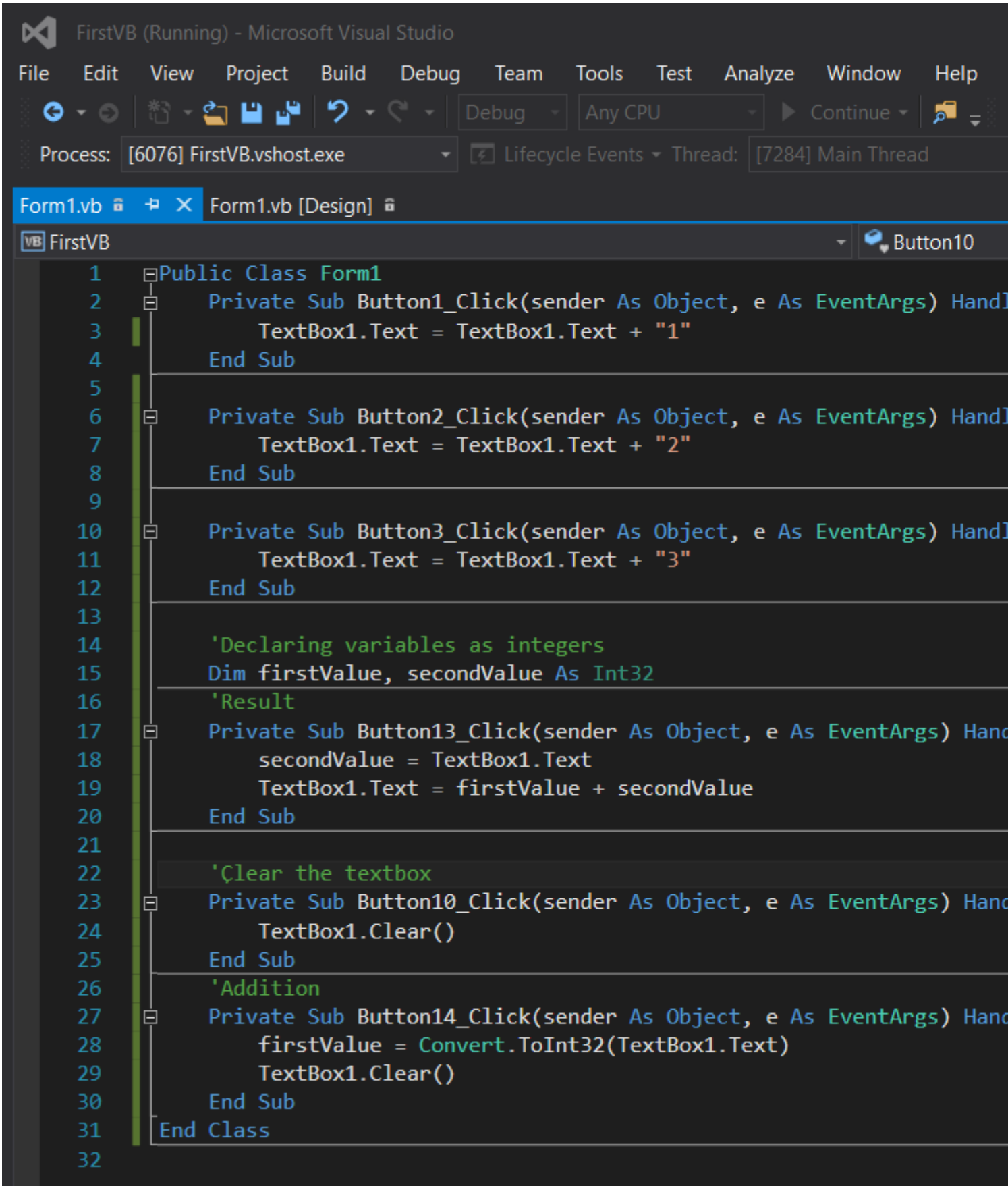
6. Familiarisez-vous avec les outils fournis dans la boîte. J'ai créé une interface de calculatrice en utilisant des boutons et une zone de texte.



7. Cliquez sur l'onglet *Propriétés* (à droite de l'éditeur). Vous pouvez modifier la propriété *Text*

d'un bouton et la zone de texte pour les renommer. *La* propriété de *police* peut être utilisée pour modifier la police des contrôles.

8. Pour écrire l'action spécifique à un événement (par exemple en cliquant sur un bouton), double-cliquez sur le contrôle. La fenêtre de code s'ouvrira.



9. VB.Net est un langage puissant conçu pour un développement rapide. Une encapsulation et une abstraction élevées sont coûteuses. Il n'est pas nécessaire d'ajouter un *point-virgule* pour indiquer la fin d'une instruction, il n'y a pas de crochets et la plupart du temps, il corrige automatiquement la casse des alphabets.
10. Le code fourni dans l'image doit être simple à comprendre. *Dim* est le mot clé utilisé pour initialiser une variable et *new* alloue de la mémoire. Tout ce que vous tapez dans la zone de texte est de type *chaîne* par défaut. Le moulage est requis pour utiliser la valeur sous un type différent.

Profitez de votre première création dans VB.Net!

Lire Premiers pas avec le langage Visual Basic .NET en ligne: https://riptutorial.com/fr/vb-net/topic/352/premiers-pas-avec-le-langage-visual-basic--net

Chapitre 2: Accès aux données

Examples

Champ de lecture de la base de données

```
Public Function GetUserFirstName(UserName As String) As String
    Dim Firstname As String = ""

    'Specify the SQL that you want to use including a Parameter
    Dim SQL As String = "select firstname from users where username=@UserName"

    'Provide a Data Source
    Dim DBDSN As String = "Data Source=server.address;Initial Catalog=DatabaseName;Persist
Security Info=True;User ID=UserName;Password=UserPassword"

    Dim dbConn As New SqlConnection(DBDSN)

    Dim dbCommand As New SqlCommand(SQL, dbConn)

    'Provide one or more Parameters
    dbCommand.Parameters.AddWithValue("@UserName", UserName)

    'An optional Timeout
    dbCommand.CommandTimeout = 600

    Dim reader As SqlDataReader
    Dim previousConnectionState As ConnectionState = dbConn.State
    Try
        If dbConn.State = ConnectionState.Closed Then
            dbConn.Open()
        End If
        reader = dbCommand.ExecuteReader
        Using reader
            With reader
                If .HasRows Then
                    'Read the 1st Record
                    reader.Read()
                    'Read required field/s
                    Firstname = .Item("FirstName").ToString
                End If

            End With

        End Using

    Catch
        'Handle the error here
    Finally
        If previousConnectionState = ConnectionState.Closed Then
            dbConn.Close()
        End If
        dbConn.Dispose()
        dbCommand.Dispose()

    End Try
    'Pass the data back from the function
```

```
    Return Firstname

End Function
```

Utiliser la fonction ci-dessus est simplement:

```
   Dim UserFirstName as string=GetUserFirstName(UserName)
```

Fonction simple à lire à partir de la base de données et à retourner en tant que DataTable

Cette fonction simple exécute la commande Select SQL spécifiée et renvoie le résultat sous forme de jeu de données.

```
Public Function ReadFromDatabase(ByVal DBConnectionString As String, ByVal SQL As String) As
DataTable
    Dim dtReturn As New DataTable
    Try
        'Open the connection using the connection string
        Using conn As New SqlClient.SqlConnection(DBConnectionString)
            conn.Open()

            Using cmd As New SqlClient.SqlCommand()
                cmd.Connection = conn
                cmd.CommandText = SQL
                Dim da As New SqlClient.SqlDataAdapter(cmd)
                da.Fill(dtReturn)
            End Using
        End Using
    Catch ex As Exception
        'Handle the exception
    End Try


    'Return the result data set
    Return dtReturn
End Function
```

Maintenant, vous pouvez exécuter la fonction ci-dessus à partir des codes ci-dessous

```
Private Sub MainFunction()
    Dim dtCustomers As New DataTable
    Dim dtEmployees As New DataTable
    Dim dtSuppliers As New DataTable


    dtCustomers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Customers]")
    dtEmployees = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Employees]")
    dtSuppliers = ReadFromDatabase("Server=MYDEVPC\SQLEXPRESS;Database=MyDatabase;User
Id=sa;Password=pwd22;", "Select * from [Suppliers]")

End Sub
```

L'exemple ci-dessus prévoit que votre instance SQL Express "SQLEXPRESS" est actuellement installée sur "MYDEVPC" et que votre base de données "MyDatabase" contient les tables "Clients", "Fournisseurs" et "Employés" et que le mot de passe "sa" est "pwd22". Veuillez modifier ces valeurs selon votre configuration pour obtenir les résultats souhaités.

Obtenir des données scalaires

Cette fonction simple peut être utilisée pour obtenir la valeur d'exactement un champ un résultat de requête d'enregistrement

```
Public Function getDataScalar(ssql As String)
    openConnection()

    Try
        Dim q As New MySqlCommand

        q.Connection = db
        q.CommandText = ssql
        getDataScalar = q.ExecuteScalar

    Catch ex As Exception
        'Exception
    End Try
End Function
```

Comment l'utiliser:

```
Dim userid as String = getDataScalar("select username from user where userid=99")
```

La variable "nom d'utilisateur" serait remplie avec la valeur du nom d'utilisateur du champ résultant de cette requête.

Lire Accès aux données en ligne: https://riptutorial.com/fr/vb-net/topic/3113/acces-aux-donnees

Chapitre 3: au hasard

Introduction

La classe Random est utilisée pour générer des entiers pseudo-aléatoires non négatifs qui ne sont pas vraiment aléatoires, mais pour des besoins généraux suffisamment proches.

La séquence est calculée à l'aide d'un numéro initial (appelé **Seed**). Dans les versions antérieures de .net, ce numéro de départ était le même chaque fois qu'une application était exécutée. Donc, ce qui arriverait, c'était que vous obteniez la même séquence de nombres pseudo-aléatoires chaque fois que l'application était exécutée. Maintenant, la graine est basée sur l'heure à laquelle l'objet est déclaré.

Remarques

Enfin, une note sur la randomisation. Comme mentionné précédemment, lorsque vous déclarez une instance de `Random` sans aucun paramètre, le constructeur utilisera l'heure actuelle dans le cadre du calcul pour créer le numéro de départ initial. Normalement c'est OK.

Toutefois. Si vous déclarez de nouvelles instances sur un très court laps de temps, chaque fois que le numéro de départ est calculé, l'heure peut être la même. Considérez ce code.

```
For i As Integer = 1 To 100000
    Dim rnd As New Random
    x = rnd.Next
Next
```

Comme les ordinateurs sont très rapides ces jours-ci, l'exécution de ce code prend une fraction de seconde et, sur plusieurs itérations séquentielles de la boucle, l'heure du système n'a pas changé. Ainsi, le numéro de graine ne changera pas et le nombre aléatoire sera le même. Si vous voulez générer beaucoup de nombres aléatoires, déclarez l'instance aléatoire en dehors de la boucle dans cet exemple simple.

```
Dim rnd As New Random
For i As Integer = 1 To 100000
    x = rnd.Next
Next
```

La règle de base est de ne pas ré-instancier le générateur de nombres aléatoires sur de courtes périodes.

Examples

Déclarer une instance

```
Dim rng As New Random()
```

Cela déclare une instance de la classe aléatoire appelée `rng` . Dans ce cas, l'heure actuelle au point où l'objet est créé est utilisée pour calculer la graine. C'est l'usage le plus courant, mais a ses propres problèmes, comme nous le verrons plus tard dans les remarques.

Au lieu de permettre au programme d'utiliser l'heure actuelle dans le cadre du calcul du numéro de départ initial, vous pouvez spécifier le numéro de départ initial. Cela peut être n'importe quel entier, constante ou variable de 32 bits. Voir ci-dessous pour des exemples. Cela signifie que votre instance va générer la même séquence de nombres pseudo-aléatoires, ce qui peut être utile dans certaines situations.

```
Dim rng As New Random(43352)
```

ou

```
Dim rng As New Random(x)
```

où `x` a été déclaré ailleurs dans votre programme en tant que constante ou variable entière.

Générer un nombre aléatoire à partir d'une instance de aléatoire

L'exemple suivant déclare une nouvelle instance de la classe Random, puis utilise la méthode `.Next` pour générer le nombre suivant dans la séquence de nombres pseudo-aléatoires.

```
Dim rnd As New Random
Dim x As Integer
x = rnd.Next
```

La dernière ligne ci-dessus générera le numéro pseudo-aléatoire suivant et l'assignera à `x` . Ce nombre sera compris entre 0 et 2147483647. Cependant, vous pouvez également spécifier la plage de nombres à générer, comme dans l'exemple ci-dessous.

```
x = rnd.Next(15, 200)
```

Veuillez noter toutefois qu'en utilisant ces paramètres, la plage de numéros sera comprise entre 15 ou plus et 199 ou moins.

Vous pouvez également générer des nombres à virgule flottante du type Double en utilisant `.NextDouble` par exemple

```
Dim rnd As New Random
Dim y As Double
y = rnd.NextDouble()
```

Vous ne pouvez toutefois pas spécifier une plage pour cela. Il sera toujours compris entre 0,0 et moins de 1,0.

Lire au hasard en ligne: https://riptutorial.com/fr/vb-net/topic/10128/au-hasard

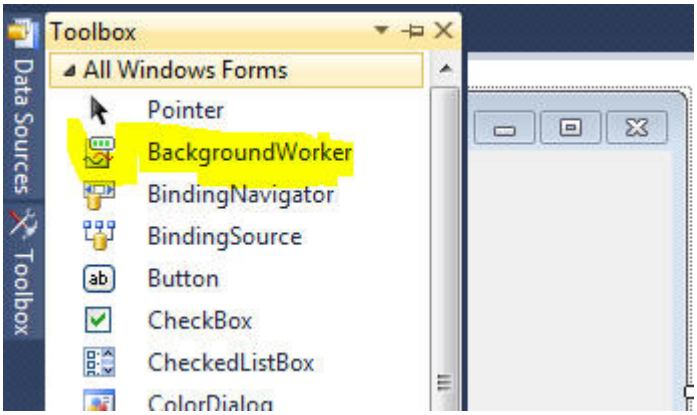
Chapitre 4: BackgroundWorker

Examples

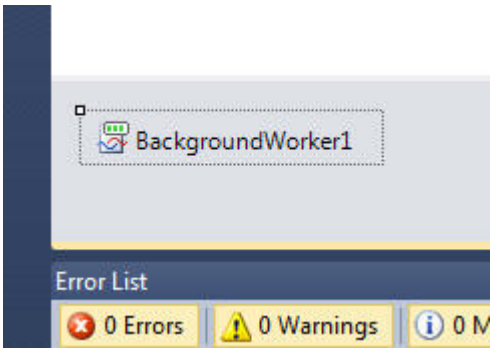
Utiliser BackgroundWorker

Exécuter une tâche avec le travailleur d'arrière-plan.

Double-cliquez sur le contrôle `BackgroundWorker` partir de la boîte à outils



Voici comment le BackgroundWorker apparaît après l'ajout.



Double-cliquez sur le contrôle ajouté pour obtenir l'événement `BackgroundWorker1_DoWork` et ajoutez le code à exécuter lorsque vous appelez BackgroundWorker. Quelque chose comme ça:

```
Private Sub BackgroundWorker1_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork

    'Do the time consuming background task here

End Sub
```

L'appel de BackgroundWorker pour effectuer la tâche peut être effectué à tout événement comme `Button_Click` , `Textbox_TextChanged` , etc. comme suit:

```
BackgroundWorker1.RunWorkerAsync()
```

Modifiez l'événement `RunWorkerCompleted` pour capturer l'événement de tâche terminé de BackgroundWorker comme suit:

```
Private Sub BackgroundWorker1_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
BackgroundWorker1.RunWorkerCompleted
    MsgBox("Done")
End Sub
```

Cela affichera une boîte de message indiquant `Done` lorsque le travailleur termine la tâche qui lui est assignée.

Accès aux composants de l'interface graphique dans BackgroundWorker

Vous ne pouvez pas accéder aux composants de l'interface graphique depuis BackgroudWorker. Par exemple, si vous essayez de faire quelque chose comme ça

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    TextBox1.Text = "Done"
End Sub
```

vous recevrez une erreur d'exécution indiquant que "l'opération cross-thread n'est pas valide: contrôlez 'TextBox1' accessible à partir d'un thread autre que celui sur lequel il a été créé."

Cela est dû au fait que BackgroundWorker exécute votre code sur un autre thread en parallèle avec le thread principal et que les composants de l'interface graphique ne sont pas compatibles avec les threads. Vous devez définir votre code à exécuter sur le thread principal à l'aide de la méthode `Invoke` , en lui attribuant un délégué:

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.Invoke(New MethodInvoker(Sub() Me.TextBox1.Text = "Done"))
End Sub
```

Ou vous pouvez utiliser la méthode ReportProgress du BackgroundWorker:

```
Private Sub BackgroundWorker1_DoWork(sender As Object, e As DoWorkEventArgs)
    Me.BackgroundWorker1.ReportProgress(0, "Done")
End Sub

Private Sub BackgroundWorker1_ProgressChanged(sender As Object, e As ProgressChangedEventArgs)
    Me.TextBox1.Text = DirectCast(e.UserState, String)
End Sub
```

Lire BackgroundWorker en ligne: https://riptutorial.com/fr/vb-net/topic/6242/backgroundworker

Chapitre 5: Compression de fichiers / dossiers

Examples

Créer une archive zip à partir du répertoire

```
System.IO.Compression.ZipFile.CreateFromDirectory("myfolder", "archive.zip")
```

Créez le fichier archive.zip contenant les fichiers qui se trouvent dans `myfolder` . Dans l'exemple, les chemins sont relatifs au répertoire de travail du programme. Vous pouvez spécifier des chemins absolus.

Extraction de l'archive zip dans le répertoire

```
System.IO.Compression.ZipFile.ExtractToDirectory("archive.zip", "myfolder")
```

Extrait archive.zip dans le répertoire myfolder. Dans l'exemple, les chemins sont relatifs au répertoire de travail du programme. Vous pouvez spécifier des chemins absolus.

Créer une archive zip dynamique

```
' Create filestream to file
Using fileStream = New IO.FileStream("archive.zip", IO.FileMode.Create)
    ' open zip archive from stream
    Using archive = New System.IO.Compression.ZipArchive(fileStream,
IO.Compression.ZipArchiveMode.Create)
        ' create file_in_archive.txt in archive
        Dim zipfile = archive.CreateEntry("file_in_archive.txt")

        ' write Hello world to file_in_archive.txt in archive
        Using sw As New IO.StreamWriter(zipfile.Open())
            sw.WriteLine("Hello world")
        End Using

    End Using
End Using
```

Ajout de la compression de fichier à votre projet

1. Dans l' *Explorateur de solutions,* accédez à votre projet, cliquez avec le bouton droit sur *Références,* puis *ajoutez une référence…*
2. Recherchez Compression et sélectionnez *System.IO.Compression.FileSystem,* puis appuyez sur OK.
3. Ajoutez `Imports System.IO.Compression` au début de votre fichier de code (avant toute classe ou module, avec les autres instructions `Imports`).

```
Option Explicit On
Option Strict On

Imports System.IO.Compression

Public Class Foo

    ...

End Class
```

Veuillez noter que cette classe (ZipArchive) n'est disponible qu'à partir de la version 4.5 de .NET

Lire Compression de fichiers / dossiers en ligne: https://riptutorial.com/fr/vb-net/topic/4638/compression-de-fichiers---dossiers

Chapitre 6: Conditions

Examples

SI ... Alors ... Sinon

```
Dim count As Integer = 0
Dim message As String

If count = 0 Then
    message = "There are no items."
ElseIf count = 1 Then
    message = "There is 1 item."
Else
    message = "There are " & count & " items."
End If
```

Si opérateur

9.0

```
If(condition > value, "True", "False")
```

Nous pouvons utiliser l'opérateur **If** au lieu de **If ... Then ... Else..End If** bloque les instructions.

Prenons l'exemple suivant:

```
If 10 > 9 Then
     MsgBox("True")
Else
     MsgBox("False")
End If
```

est le même que

```
MsgBox(If(10 > 9, "True", "False"))
```

`If()` utilise l'évaluation de *court-circuit* , ce qui signifie qu'il n'évaluera que les arguments qu'il utilise. Si la condition est fausse (ou `Nullable` qui est `Nothing`), la première alternative ne sera pas évaluée du tout et aucun de ses effets secondaires ne sera observé. Ceci est effectivement le même que l' opérateur ternaire de C # sous la forme de `condition?a:b` .

Ceci est particulièrement utile pour éviter les exceptions:

```
Dim z As Integer = If(x = 0, 0, y/x)
```

Nous savons tous que la division par zéro lancera une exception, mais `If()` protège contre cela en court-circuitant uniquement l'expression que la condition a déjà assurée est valide.

Un autre exemple:

```
Dim varDate as DateTime = If(varString <> "N/A", Convert.ToDateTime(varString), Now.Date)
```

Si `varString <> "N/A"` évalue à `False` , il attribuera `varDate` valeur de comme `Now.Date` sans évaluer la première expression.

9.0

Les anciennes versions de VB n'ont pas l'opérateur `If()` et doivent se débrouiller avec la fonction `IIf()` . Comme il est une fonction, pas un opérateur, il *ne* court-circuit; toutes les expressions sont évaluées, avec tous les effets secondaires possibles, y compris les pénalités de performance, le changement d'état et les exceptions de lancement. (Les deux exemples ci-dessus qui évitent les exceptions seraient `IIf` s'ils étaient convertis en `IIf` .) Si l'un de ces effets secondaires présente un problème, il est impossible d'utiliser un conditionnel en ligne. au lieu de cela, compter sur `If..Then` bloque comme d'habitude.

Lire Conditions en ligne: https://riptutorial.com/fr/vb-net/topic/7484/conditions

Chapitre 7: Console

Examples

Console.ReadLine ()

```
Dim input as String = Console.ReadLine()
```

`Console.ReadLine()` lira l'entrée de la console depuis l'utilisateur jusqu'à ce que la nouvelle ligne soit détectée (en général, en appuyant sur la touche Entrée ou Retour). L'exécution du code est suspendue dans le thread en cours jusqu'à ce qu'une nouvelle ligne soit fournie. Ensuite, la ligne de code suivante sera exécutée.

Console.WriteLine ()

```
Dim x As Int32 = 128
Console.WriteLine(x) ' Variable '
Console.WriteLine(3) ' Integer '
Console.WriteLine(3.14159) ' Floating-point number '
Console.WriteLine("Hello, world") ' String '
Console.WriteLine(myObject) ' Outputs the value from calling myObject.ToString()
```

La méthode `Console.WriteLine()` imprimera le ou les arguments donnés **avec** une nouvelle ligne attachée à la fin. Cela imprimera tout objet fourni, y compris, mais sans s'y limiter, les chaînes, les entiers, les variables, les nombres à virgule flottante.

Lors de l'écriture d'objets qui ne sont pas explicitement appelés par les diverses surcharges `WriteLine` (c'est-à-dire que vous utilisez la surcharge qui attend une valeur de type `Object` , WriteLine utilisera la méthode `.ToString()` pour générer une `String` pour écrire réellement). Les objets doivent remplacer la méthode `.ToString` et produire quelque chose de plus significatif que l'implémentation par défaut (qui écrit généralement uniquement le nom complet du type).

Console.Write ()

```
Dim x As Int32 = 128
Console.Write(x) ' Variable '
Console.Write(3) ' Integer '
Console.Write(3.14159) ' Floating-point number '
Console.Write("Hello, world") ' String '
```

La méthode `Console.Write()` est identique à la méthode `Console.WriteLine()` sauf qu'elle imprime le ou les arguments donnés **sans** nouvelle ligne à la fin. Cette méthode peut être rendue fonctionnellement identique à `WriteLine` en ajoutant une chaîne de nouvelle ligne à la fin des arguments fournis:

```
Console.Write("this is the value" & Environment.NewLine)
```

Console.Read ()

```
Dim inputCode As Integer = Console.Read()
```

`Console.Read()` attend l'entrée de l'utilisateur et, à sa réception, renvoie une valeur entière correspondant au code de caractère du caractère saisi. Si le flux d'entrée est terminé d'une certaine manière avant que l'entrée puisse être obtenue, -1 est renvoyé à la place.

Console.ReadKey ()

```
Dim inputChar As ConsoleKeyInfo = Console.ReadKey()
```

`Console.ReadKey()` attend l'entrée de l'utilisateur et, à sa réception, renvoie un objet de la classe `ConsoleKeyInfo` , qui contient des informations relatives au caractère fourni par l'utilisateur. Pour plus de détails concernant les informations fournies, visitez la documentation MSDN .

Prototype de l'invite de ligne de commande

```
Module MainPrompt
Public Const PromptSymbol As String = "TLA > "
Public Const ApplicationTitle As String = GetType(Project.BaseClass).Assembly.FullName
REM Or you can use a custom string
REM Public Const ApplicationTitle As String = "Short name of the application"

Sub Main()
    Dim Statement As String
    Dim BrokenDownStatement As String()
    Dim Command As String
    Dim Args As String()
    Dim Result As String

    Console.ForegroundColor = ConsoleColor.Cyan
    Console.Title = ApplicationTitle & " command line console"

    Console.WriteLine("Welcome to " & ApplicationTitle & "console frontend")
    Console.WriteLine("This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString)
    Console.WriteLine()
    Console.Write(PromptSymbol)

    Do While True
        Statement = Console.ReadLine()
        BrokenDownStatement = Statement.Split(" ")
        ReDim Args(BrokenDownStatement.Length - 1)
        Command = BrokenDownStatement(0)

        For i = 1 To BrokenDownStatement.Length - 1
            Args(i - 1) = BrokenDownStatement(i)
        Next

        Select Case Command.ToLower
            Case "example"
                Result = DoSomething(Example)
            Case "exit", "quit"
```

```
                Exit Do
            Case "ver"
                Result = "This package is version " &
GetType(Project.BaseClass).Assembly.GetName().Version.ToString
            Case Else
                Result = "Command not acknowldged: -" & Command & "-"
        End Select
        Console.WriteLine(" " & Result)
        Console.Write(PromptSymbol)
    Loop

    Console.WriteLine("I am exiting, time is " & DateTime.Now.ToString("u"))
    Console.WriteLine("Goodbye")
    Environment.Exit(0)
End Sub
End Module
```

Ce prototype génère un interpréteur de ligne de commande de base.

Il obtient automatiquement le nom de l'application et la version à communiquer à l'utilisateur. Pour chaque ligne d'entrée, il reconnaît la commande et une liste arbitraire d'arguments, tous séparés par un espace.

Comme exemple de base, ce code comprend les commandes *ver* , *quit* et *exit* .

Le paramètre *Project.BaseClass* est une classe de votre projet où les détails d'assemblage sont définis.

Lire Console en ligne: https://riptutorial.com/fr/vb-net/topic/602/console

Chapitre 8: Conversion de type

Syntaxe

- CBool (expression)
- CByte (expression)
- CChar (expression)
- CDate (expression)
- CDbl (expression)
- CDec (expression)
- CInt (expression)
- CLng (expression)
- CObj (expression)
- CSByte (expression)
- CShort (expression)
- CSng (expression)
- CStr (expression)
- CUInt (expression)
- CULng (expression)
- CUShort (expression)

Paramètres

Nom de la fonction	Argument de plage pour l'expression
CBool	Toute expression ou chaîne valide ou une expression numérique
CByte	0 à 255 (non signé); les parties fractionnaires sont arrondies.
CChar	Toute expression Char ou String valide; seul le premier caractère d'une chaîne est converti; La valeur peut être comprise entre 0 et 65535 (non signé).

Examples

Conversion du texte de la zone de texte en un entier

À partir de MSDN

> Utilisez la fonction CInt pour fournir des conversions de tout autre type de données à un sous-type entier. Par exemple, CInt force l'arithmétique entière lorsque des opérations arithmétiques en devise, en simple précision ou en double précision ont lieu.

En supposant que vous avez 1 bouton et 2 zone de texte. Si vous tapez sur textbox1.text `5.5` et sur textbox2.text `10` .

Si vous avez ce code:

```
Dim result = textbox1.text + textbox2.text
MsgBox("Result: " & result)
'It will output
5.510
```

Pour ajouter les valeurs des 2 zones de texte, vous devez convertir leurs valeurs en `Int` à l'aide de `CInt(expression)` .

```
Dim result = CInt(textbox1.text) + CInt(textbox2.text)
MsgBox("Result: " & result)
'It will output
16
```

> Remarque: Lorsque la partie fractionnaire d'une valeur est exactement égale à 0,5, la fonction CInt arrondit au nombre pair le plus proche. Par exemple, **0,5 arrondit à 0** , tandis que **1,5 arrondit à 2 et 3,5 à 4** . L'arrondi au nombre pair le plus proche a pour but de compenser un biais qui pourrait s'accumuler lorsque de nombreux nombres sont additionnés.

Lire Conversion de type en ligne: https://riptutorial.com/fr/vb-net/topic/4681/conversion-de-type

Chapitre 9: Déboguer votre application

Introduction

Chaque fois que vous rencontrez un problème dans votre code, il est toujours utile de savoir ce qui se passe à l'intérieur. La classe System.Diagnostics.Debug dans .Net Framework vous aidera beaucoup dans cette tâche.

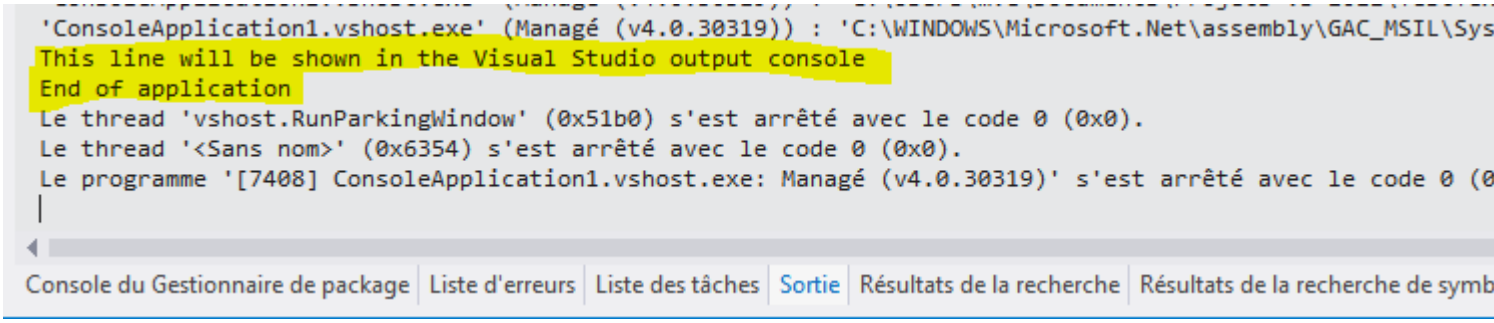
Le premier avantage de la classe Debug est qu'elle ne produit du code que si vous créez votre application en mode Debug. Lorsque vous créez votre application en mode Release, aucun code ne sera généré à partir des appels Debug.

Examples

Debug dans la console

```
Module Module1
    Sub Main()
        Debug.WriteLine("This line will be shown in the Visual Studio output console")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
End Module
```

produira:



Indentation de votre sortie de débogage

```
Module Module1

    Sub Main()
        Debug.WriteLine("Starting aplication")

        Debug.Indent()
        LoopAndDoStuff(5)
        Debug.Unindent()
```

```
        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub

    Sub LoopAndDoStuff(Iterations As Integer)
        Dim x As Integer = 0
        Debug.WriteLine("Starting loop")
        Debug.Indent()
        For i As Integer = 0 To Iterations - 1
            Debug.Write("Iteration " & (i + 1).ToString() & " of " & Iterations.ToString() &
": Value of X: ")
            x += (x + 1)
            Debug.WriteLine(x.ToString())
        Next
        Debug.Unindent()
        Debug.WriteLine("Loop is over")
    End Sub
End Module
```

produira:

```
'ConsoleApplication1.vshost.exe' (Managé (v4.0.30319)) : 'C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\Syster
Starting aplication
    Starting loop
        Iteration 1 of 5: Value of X: 1
        Iteration 2 of 5: Value of X: 3
        Iteration 3 of 5: Value of X: 7
        Iteration 4 of 5: Value of X: 15
        Iteration 5 of 5: Value of X: 31
    Loop is over
End of application
Le thread 'vshost.RunParkingWindow' (0x2764) s'est arrêté avec le code 0 (0x0).
Le thread '<Sans nom>' (0xe74) s'est arrêté avec le code 0 (0x0).
Le programme '[8316] ConsoleApplication1.vshost.exe: Managé (v4.0.30319)' s'est arrêté avec le code 0 (0x0
```

onsole du Gestionnaire de package | Liste d'erreurs | Liste des tâches | Sortie | Résultats de la recherche | Résultats de la recherche de symbole

Déboguer dans un fichier texte

Au début de votre application, vous devez ajouter un TextWriterTraceListener à la liste Listeners de la classe Debug.

```
Module Module1

    Sub Main()
        Debug.Listeners.Add(New TextWriterTraceListener("Debug of " & DateTime.Now.ToString()
& ".txt"))

        Debug.WriteLine("Starting aplication")

        Console.WriteLine("Press a key to exit")
        Console.ReadKey()

        Debug.WriteLine("End of application")
    End Sub
```

```
End Module
```

Tout le code de débogage produit sera affiché dans la console Visual Studio ET dans le fichier texte que vous avez choisi.

Si le fichier est toujours le même:

```
Debug.Listeners.Add(New TextWriterTraceListener("Debug.txt"))
```

Le résultat sera ajouté au fichier à chaque fois ET un nouveau fichier commençant par un GUID, puis votre nom de fichier sera généré.

Lire Déboguer votre application en ligne: https://riptutorial.com/fr/vb-net/topic/8631/deboguer-votre-application

Chapitre 10: Déclaration de variables

Syntaxe

- Compteur public As Integer
- Compteur privé comme entier
- Compteur de comptage As Integer

Examples

Déclarer et assigner une variable en utilisant un type primitif

Les variables dans Visual Basic sont déclarées à l'aide du mot clé `Dim` . Par exemple, cela déclare une nouvelle variable appelée `counter` avec le type de données `Integer` :

```
Dim counter As Integer
```

Une déclaration de variable peut également inclure un modificateur d'accès , tel que `Public` , `Protected` , `Friend` ou `Private` . Cela fonctionne conjointement avec la portée de la variable pour déterminer son accessibilité.

Modificateur d'accès	Sens
Publique	Tous les types pouvant accéder au type englobant
Protégé	Seule la classe englobante et celles qui en héritent
Ami	Tous les types d'un même assemblage pouvant accéder au type englobant
Ami protégé	La classe englobante et ses héritiers, *ou* les types du même assembly pouvant accéder à la classe englobante
Privé	Seul le type de clôture
Statique	Seulement sur les variables locales et ne s'initialise qu'une seule fois.

En tant que raccourci, le mot-clé `Dim` peut être remplacé par le modificateur d’accès dans la déclaration de la variable:

```
Public TotalItems As Integer
Private counter As Integer
```

Les types de données pris en charge sont décrits dans le tableau ci-dessous:

Type	Alias	Allocation de mémoire	Exemple
SByte	N / A	Un octet	`Dim example As SByte = 10`
Int16	Court	2 bytes	`Dim example As Short = 10`
Int32	Entier	4 octets	`Dim example As Integer = 10`
Int64	Longue	8 octets	`Dim example As Long = 10`
Unique	N / A	4 octets	`Dim example As Single = 10.95`
Double	N / A	8 octets	`Dim example As Double = 10.95`
Décimal	N / A	16 octets	`Dim example As Decimal = 10.95`
Booléen	N / A	Dicté par la mise en œuvre de la plate-forme	`Dim example As Boolean = True`
Carboniser	N / A	2 octets	`Dim example As Char = "A"C`
Chaîne	N / A	$20 + \left\lfloor \frac{length}{2} \right\rfloor * 4bytes$ la source	`Dim example As String = "Stack Overflow"`
DateTime	Rendez-vous amoureux	8 octets	`Dim example As Date = Date.Now`
Octet	N / A	1 octet	`Dim example As Byte = 10`
UInt16	UShort	2 octets	`Dim example As UShort = 10`
UInt32	UInteger	4 octets	`Dim example As UInteger = 10`
Utt64	ULong	8 octets	`Dim example As ULong = 10`
Objet	N / A	Architecture 32 bits 4 octets, architecture 64 bits 8 octets	`Dim example As Object = Nothing`

Il existe également des caractères de type identifiant et littéral utilisables en remplacement du type textuel et / ou pour forcer le type littéral:

Type (ou alias)	Caractère de type d'identifiant	Caractère de type littéral
Court	N / A	`example = 10S`
Entier	`Dim example%`	`example = 10%` ou `example = 10I`

Type (ou alias)	Caractère de type d'identifiant	Caractère de type littéral
Longue	`Dim example&`	`example = 10&` ou `example = 10L`
Unique	`Dim example!`	`example = 10!` ou `example = 10F`
Double	`Dim example#`	`example = 10#` ou `example = 10R`
Décimal	`Dim example@`	`example = 10@` ou `example = 10D`
Carboniser	N / A	`example = "A"C`
Chaîne	`Dim example$`	N / A
UShort	N / A	`example = 10US`
UInteger	N / A	`example = 10UI`
ULong	N / A	`example = 10UL`

Les suffixes intégraux sont également utilisables avec des préfixes hexadécimaux (& H) ou octaux (& O):

`example = &H8000S` ou `example = &O77&`

Les objets de date (heure) peuvent également être définis en utilisant une syntaxe littérale:

```
Dim example As Date = #7/26/2016 12:8 PM#
```

Une fois qu'une variable est déclarée, elle existera dans la portée du type contenant, `Sub` - `Function` ou `Function` déclarée, à titre d'exemple:

```
Public Function IncrementCounter() As Integer
    Dim counter As Integer = 0
    counter += 1

    Return counter
End Function
```

La variable de compteur n'existera que jusqu'à la `End Function` et sera alors hors de portée. Si cette variable de compteur est nécessaire en dehors de la fonction, vous devrez la définir au niveau de la classe / structure ou du module.

```
Public Class ExampleClass

    Private _counter As Integer

    Public Function IncrementCounter() As Integer
       _counter += 1
       Return _counter
    End Function

End Class
```

Vous pouvez également utiliser le modificateur `Static` (à ne pas confondre avec `Shared`) pour

permettre à une variable locale de conserver sa valeur entre les appels de sa méthode englobante:

```
Function IncrementCounter() As Integer
    Static counter As Integer = 0
    counter += 1

    Return counter
End Function
```

Niveaux de déclaration - Variables locales et membres

Variables locales - Celles déclarées dans une procédure (sous-routine ou fonction) d'une classe (ou d'une autre structure). Dans cet exemple, `exampleLocalVariable` est une variable locale déclarée dans `ExampleFunction()` :

```
Public Class ExampleClass1

    Public Function ExampleFunction() As Integer
        Dim exampleLocalVariable As Integer = 3
        Return exampleLocalVariable
    End Function

End Class
```

Le mot-clé `Static` permet de conserver une variable locale et de conserver sa valeur après la terminaison (où généralement les variables locales cessent d'exister à la fin de la procédure contenant).

Dans cet exemple, la console est `024` . À chaque appel à `ExampleSub()` de `Main()` la variable statique conserve la valeur `ExampleSub()` à la fin de l'appel précédent:

```
Module Module1

    Sub Main()
        ExampleSub()
        ExampleSub()
        ExampleSub()
    End Sub

    Public Sub ExampleSub()
        Static exampleStaticLocalVariable As Integer = 0
        Console.Write(exampleStaticLocalVariable.ToString)
        exampleStaticLocalVariable += 2
    End Sub

End Module
```

Variables de membre - Déclarées en dehors de toute procédure, au niveau de la classe (ou autre structure). Il peut s'agir de **variables d'instance** , dans lesquelles chaque instance de la classe contenante possède sa propre copie distincte de cette variable ou de **variables** `Shared` , qui existent en tant que variable unique associée à la classe elle-même, indépendamment de toute instance.

Ici, `ExampleClass2` contient deux variables membres. Chaque instance du `ExampleClass2` a un individu `ExampleInstanceVariable` accessible via la référence de classe. La variable partagée `ExampleSharedVariable` est cependant accessible en utilisant le nom de la classe:

```
Module Module1

    Sub Main()

        Dim instance1 As ExampleClass4 = New ExampleClass4
        instance1.ExampleInstanceVariable = "Foo"

        Dim instance2 As ExampleClass4 = New ExampleClass4
        instance2.ExampleInstanceVariable = "Bar"

        Console.WriteLine(instance1.ExampleInstanceVariable)
        Console.WriteLine(instance2.ExampleInstanceVariable)
        Console.WriteLine(ExampleClass4.ExampleSharedVariable)

    End Sub

    Public Class ExampleClass4

        Public ExampleInstanceVariable As String
        Public Shared ExampleSharedVariable As String = "FizzBuzz"

    End Class

End Module
```

Exemple de modificateurs d'accès

Dans l'exemple suivant, considérez que vous disposez d'une solution hébergeant deux projets: **ConsoleApplication1** et **SampleClassLibrary** . Le premier projet aura les classes **SampleClass1** et **SampleClass2** . Le second aura **SampleClass3** et **SampleClass4** . En d'autres termes, nous avons deux assemblées avec deux classes chacune. **ConsoleApplication1** a une référence à **SampleClassLibrary** .

Voir comment **SampleClass1.MethodA** interagit avec d'autres classes et méthodes.

SampleClass1.vb:

```
Imports SampleClassLibrary

Public Class SampleClass1
    Public Sub MethodA()
        'MethodA can call any of the following methods because
        'they all are in the same scope.
        MethodB()
        MethodC()
        MethodD()
        MethodE()

        'Sample2 is defined as friend. It is accessible within
        'the type itself and all namespaces and code within the same assembly.
        Dim class2 As New SampleClass2()
        class2.MethodA()
```

```
        'class2.MethodB() 'SampleClass2.MethodB is not accessible because
                          'this method is private. SampleClass2.MethodB
                          'can only be called from SampleClass2.MethodA,
                          'SampleClass2.MethodC, SampleClass2.MethodD
                          'and SampleClass2.MethodE
        class2.MethodC()
        'class2.MethodD() 'SampleClass2.MethodD is not accessible because
                          'this method is protected. SampleClass2.MethodD
                          'can only be called from any class that inherits
                          'SampleClass2, SampleClass2.MethodA, SampleClass2.MethodC,
                          'SampleClass2.MethodD and SampleClass2.MethodE
        class2.MethodE()

        Dim class3 As New SampleClass3() 'SampleClass3 resides in other
                                         'assembly and is defined as public.
                                         'It is accessible anywhere.
        class3.MethodA()
        'class3.MethodB() 'SampleClass3.MethodB is not accessible because
                          'this method is private. SampleClass3.MethodB can
                          'only be called from SampleClass3.MethodA,
                          'SampleClass3.MethodC, SampleClass3.MethodD
                          'and SampleClass3.MethodE

        'class3.MethodC() 'SampleClass3.MethodC is not accessible because
                          'this method is friend and resides in another assembly.
                          'SampleClass3.MethodC can only be called anywhere from the
                          'same assembly, SampleClass3.MethodA, SampleClass3.MethodB,
                          'SampleClass3.MethodD and SampleClass3.MethodE

        'class4.MethodD() 'SampleClass3.MethodE is not accessible because
                          'this method is protected friend. SampleClass3.MethodD
                          'can only be called from any class that resides inside
                          'the same assembly and inherits SampleClass3,
                          'SampleClass3.MethodA, SampleClass3.MethodB,
                          'SampleClass3.MethodC and SampleClass3.MethodD

        'Dim class4 As New SampleClass4() 'SampleClass4 is not accessible because
                                          'it is defined as friend and resides in
                                          'other assembly.
    End Sub

    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass2.vb:

```
Friend Class SampleClass2
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub

    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass3.vb:

```
Public Class SampleClass3
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub

    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

SampleClass4.vb:

```
Friend Class SampleClass4
    Public Sub MethodA()
        'Doing MethodA stuff...
    End Sub
    Private Sub MethodB()
        'Doing MethodB stuff...
    End Sub

    Friend Sub MethodC()
        'Doing MethodC stuff...
    End Sub
```

```
    Protected Sub MethodD()
        'Doing MethodD stuff...
    End Sub

    Protected Friend Sub MethodE()
        'Doing MethodE stuff...
    End Sub
End Class
```

Lire Déclaration de variables en ligne: https://riptutorial.com/fr/vb-net/topic/3366/declaration-de-variables

Chapitre 11: Des classes

Introduction

Une classe regroupe différentes fonctions, méthodes, variables et propriétés, appelées ses membres. Une classe encapsule les membres, auxquels une instance de la classe peut accéder, appelée un objet. Les classes sont extrêmement utiles pour le programmeur, car elles rendent la tâche pratique et rapide, avec des caractéristiques telles que la modularité, la réutilisabilité, la facilité de maintenance et la lisibilité du code.

Les classes sont les blocs de construction des langages de programmation orientés objet.

Examples

Créer des classes

Les classes permettent de créer vos propres types dans le framework .NET. Dans une définition de classe, vous pouvez inclure les éléments suivants:

- Des champs
- Propriétés
- Les méthodes
- Constructeurs
- Événements

Pour déclarer une classe, utilisez la syntaxe suivante:

```
Public Class Vehicle
End Class
```

D'autres types .NET peuvent être encapsulés dans la classe et exposés en conséquence, comme indiqué ci-dessous:

```
Public Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Classes abstraites

Si les classes partagent des fonctionnalités communes, vous pouvez les regrouper dans une classe de base ou abstraite. Les classes abstraites peuvent contenir une implémentation partielle ou nulle et permettent au type dérivé de remplacer l'implémentation de base.

Les classes abstraites de VisualBasic.NET doivent être déclarées comme `MustInherit` et ne peuvent pas être instanciées.

```
Public MustInherit Class Vehicle
     Private Property _numberOfWheels As Integer
     Private Property _engineSize As Integer

     Public Sub New(engineSize As Integer, wheels As Integer)
         _numberOfWheels = wheels
         _engineSize = engineSize
     End Sub

     Public Function DisplayWheelCount() As Integer
         Return _numberOfWheels
     End Function
End Class
```

Un sous-type peut alors `inherit` cette classe abstraite comme indiqué ci-dessous:

```
Public Class Car
    Inherits Vehicle
End Class
```

Car héritera de tous les types déclarés dans le véhicule, mais ne pourra y accéder qu'en fonction du modificateur d'accès sous-jacent.

```
Dim car As New Car()
car.DisplayWheelCount()
```

Dans l'exemple ci-dessus, une nouvelle instance de voiture est créée. La méthode `DisplayWheelCount()` est alors appelée pour appeler la classe de base `DisplayWheelCount()` des `Vehicles` .

Lire Des classes en ligne: https://riptutorial.com/fr/vb-net/topic/3522/des-classes

Chapitre 12: Des listes

Syntaxe

- List.Add (article en tant que type)
- List.RemoveRange (index As Integer, compte comme Integer)
- List.Remove (index As Integer)
- List.AddRange (collection)
- List.Find (correspond à Predicate (of String))
- List.Insert (index sous forme de nombre entier, élément sous forme de type)
- List.Contains (élément en tant que type)

Examples

Créer une liste

Les listes peuvent être remplies avec n'importe quel type de données si nécessaire, avec le format

```
Dim aList as New List(Of Type)
```

Par exemple:

Créer une nouvelle liste vide de chaînes

```
Dim aList As New List(Of String)
```

Créer une nouvelle liste de chaînes et remplir certaines données

VB.NET 2005/2008:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

VB.NET 2010:

```
Dim aList as New List(Of String) From {"one", "two", "three"}
```

-

VB.NET 2015:

```
Dim aList as New List(Of String)(New String() {"one", "two", "three"})
```

REMARQUE:

Si vous recevez ce qui suit lorsque le code est exécuté:

> La référence d'objet n'est pas définie à une instance d'un objet.

Assurez-vous de déclarer en tant que `New` ie `Dim aList as New List(Of String)` ou en déclarant sans le `New` , assurez-vous de définir la liste sur une nouvelle liste - `Dim aList as List(Of String) = New List(Of String)`

Ajouter des éléments à une liste

```
Dim aList as New List(Of Integer)
aList.Add(1)
aList.Add(10)
aList.Add(1001)
```

Pour ajouter plusieurs éléments à la fois, utilisez **AddRange** . Ajoute toujours à la fin de la liste

```
Dim blist as New List(of Integer)
blist.AddRange(alist)


Dim aList as New List(of String)
alist.AddRange({"one", "two", "three"})
```

Pour ajouter des éléments au milieu de la liste, utilisez **Insérer**

Insérer place l'élément à l'index et renuméroter les éléments restants

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("three")
alist(0) = "one"
alist(1) = "three"
alist.Insert(1,"two")
```

Nouvelle sortie:

```
alist(0) = "one"
alist(1) = "two"
alist(2) = "three"
```

Supprimer des éléments d'une liste

```
Dim aList As New List(Of String)
aList.Add("Hello")
aList.Add("Delete Me!")
aList.Add("World")

'Remove the item from the list at index 1
aList.RemoveAt(1)

'Remove a range of items from a list, starting at index 0, for a count of 1)
'This will remove index 0, and 1!
```

```
aList.RemoveRange(0, 1)

'Clear the entire list
alist.Clear()
```

Récupérer des éléments d'une liste

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList(0)
```

`output` :

> Bonjour le monde

Si vous ne connaissez pas l'index de l'élément ou si vous ne connaissez qu'une partie de la chaîne, utilisez la méthode **Find** ou **FindAll**

```
Dim aList as New List(Of String)
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.Find(Function(x) x.StartWith("Hello"))
```

`output` :

> Bonjour le monde

La méthode **FindAll** renvoie une nouvelle liste (de chaîne)

```
Dim aList as New List(Of String)
aList.Add("Hello, Test")
aList.Add("Hello, World")
aList.Add("Test")

Dim output As String = aList.FindAll(Function(x) x.Contains("Test"))
```

> output (0) = "Bonjour, test"
>
> output (1) = "Test"

Boucle des éléments de la liste dans la liste

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For Each str As String in aList
    System.Console.WriteLine(str)
```

```
Next
```

Produit la sortie suivante:

```
one
two
three
```

Une autre option serait de parcourir en boucle l'index de chaque élément:

```
Dim aList as New List(Of String)
aList.Add("one")
aList.Add("two")
aList.Add("three")

For i = 0 to aList.Count - 1 'We use "- 1" because a list uses 0 based indexing.
    System.Console.WriteLine(aList(i))
Next
```

Vérifier si l'élément existe dans une liste

```
    Sub Main()
        Dim People = New List(Of String)({"Bob Barker", "Ricky Bobby", "Jeff Bridges"})
        Console.WriteLine(People.Contains("Rick James"))
        Console.WriteLine(People.Contains("Ricky Bobby"))
        Console.WriteLine(People.Contains("Barker"))
        Console.Read
    End Sub
```

Produit la sortie suivante:

```
False
True
False
```

Lire Des listes en ligne: https://riptutorial.com/fr/vb-net/topic/3636/des-listes

Chapitre 13: Dictionnaires

Introduction

Un dictionnaire représente une collection de clés et de valeurs. Voir la classe Dictionnaire MSDN (Tkey, TValue) .

Examples

Traverser un dictionnaire et imprimer toutes les entrées

Chaque paire du dictionnaire est une instance de `KeyValuePair` avec les mêmes paramètres de type que le dictionnaire. Lorsque vous parcourez le dictionnaire avec `For Each` , chaque itération vous donnera l'une des paires clé-valeur stockées dans le dictionnaire.

```
For Each kvp As KeyValuePair(Of String, String) In currentDictionary
  Console.WriteLine("{0}: {1}", kvp.Key, kvp.Value)
Next
```

Créer un dictionnaire rempli de valeurs

```
Dim extensions As New Dictionary(Of String, String) _
  from { { "txt", "notepad" },
  { "bmp", "paint" },
  { "doc", "winword" } }
```

Cela crée un dictionnaire et le remplit immédiatement avec trois KeyValuePairs.

Vous pouvez également ajouter de nouvelles valeurs ultérieurement en utilisant la méthode Add:

```
extensions.Add("png", "paint")
```

Notez que la clé (le premier paramètre) doit être unique dans le dictionnaire, sinon une exception sera lancée.

Obtenir une valeur de dictionnaire

Vous pouvez obtenir la valeur d'une entrée dans le dictionnaire en utilisant la propriété 'Item':

```
Dim extensions As New Dictionary(Of String, String) From {
    { "txt", "notepad" },
    { "bmp", "paint" },
    { "doc", "winword" }
}

Dim program As String = extensions.Item("txt") 'will be "notepad"
```

```
' alternative syntax as Item is the default property (a.k.a. indexer)
Dim program As String = extensions("txt") 'will be "notepad"

' other alternative syntax using the (rare)
' dictionary member access operator (a.k.a. bang operator)
Dim program As String = extensions!txt 'will be "notepad"
```

Si la clé n'est pas présente dans le dictionnaire, une exception KeyNotFoundException sera lancée.

Vérification de la clé déjà dans le dictionnaire - réduction des données

La méthode `ConstainsKey` est le moyen de savoir si une clé existe déjà dans le dictionnaire.

Cela est pratique pour la réduction des données. Dans l'exemple ci-dessous, chaque fois que nous rencontrons un nouveau mot, nous l'ajoutons en tant que clé dans le dictionnaire, sinon nous incrémentons le compteur pour ce mot spécifique.

```
Dim dic As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)

Dim words As String() = Split(<big text source>," ", -1, CompareMethod.Binary)

For Each str As String In words
    If dic.ContainsKey(str) Then
        dic(str) += 1
    Else
        dic.Add(str, 1)
    End If
Next
```

Exemple de réduction XML: obtenir tous les noms de nœuds enfants et leur occurrence dans une branche d'un document XML

```
Dim nodes As IDictionary(Of String, Integer) = New Dictionary(Of String, Integer)
Dim xmlsrc = New XmlDocument()
xmlsrc.LoadXml(<any text stream source>)

For Each xn As XmlNode In xmlsrc.FirstChild.ChildNodes 'selects the proper parent
    If nodes.ContainsKey(xn.Name) Then
        nodes(xn.Name) += 1
    Else
        nodes.Add(xn.Name, 1)
    End If
Next
```

Lire Dictionnaires en ligne: https://riptutorial.com/fr/vb-net/topic/2165/dictionnaires

Chapitre 14: En boucle

Examples

Pour ... Suivant

`Next` boucle `For` ... `Next` est utilisée pour répéter la même action pour un nombre fini de fois. Les instructions à l'intérieur de la boucle suivante seront exécutées 11 fois. La première fois, `i` la valeur 0, la deuxième fois la valeur 1, la dernière fois 10.

```
For i As Integer = 0 To 10
    'Execute the action
    Console.Writeline(i.ToString)
Next
```

Toute expression entière peut être utilisée pour paramétrer la boucle. Il est permis, mais pas obligatoire, que la variable de contrôle (dans ce cas `i`) soit également indiquée après la `Next` . Il est permis que la variable de contrôle soit déclarée à l'avance, plutôt que dans l'instruction `For` .

```
Dim StartIndex As Integer = 3
Dim EndIndex As Integer = 7
Dim i As Integer

For i = StartIndex To EndIndex - 1
    'Execute the action
    Console.Writeline(i.ToString)
Next i
```

La possibilité de définir les entiers de début et de fin permet de créer des boucles qui référencent directement d'autres objets, telles que:

```
For i = 0 to DataGridView1.Rows.Count - 1
    Console.Writeline(DataGridView1.Rows(i).Cells(0).Value.ToString)
Next
```

Cela ferait alors une boucle sur chaque ligne de `DataGridView1` et effectuerait l'action d'écrire la valeur de la colonne 1 dans la console. *(Le -1 est parce que la première ligne des lignes comptées serait 1, pas 0)*

Il est également possible de définir comment la variable de contrôle doit s'incrémenter.

```
For i As Integer = 1 To 10 Step 2
    Console.Writeline(i.ToString)
Next
```

Cela produit:

    1 3 5 7 9

Il est également possible de décrémenter la variable de contrôle (décompte).

```
For i As Integer = 10 To 1 Step -1
    Console.Writeline(i.ToString)
Next
```

Cela produit:

    10 9 8 7 6 5 4 3 2 1

Vous ne devez pas essayer d'utiliser (lire ou mettre à jour) la variable de contrôle en dehors de la boucle.

For Each ... Prochaine boucle pour boucler la collection d'éléments

Vous pouvez utiliser une boucle `For Each...Next` pour parcourir tout type `IEnumerable` . Cela inclut les tableaux, les listes et tout ce qui peut être de type IEnumerable ou renvoie un IEnumerable.

Un exemple de boucle à travers une propriété Rows de DataTable ressemblerait à ceci:

```
For Each row As DataRow In DataTable1.Rows
    'Each time this loops, row will be the next item out of Rows
    'Here we print the first column's value from the row variable.
    Debug.Print(Row.Item(0))
Next
```

Une chose importante à noter est que la collection ne doit pas être modifiée dans une boucle `For Each` . Cela entraînera une `System.InvalidOperationException` avec le message:

    La collection a été modifiée; l'opération d'énumération peut ne pas s'exécuter.

Boucle while à itérer alors qu'une condition est vraie

Une boucle `While` commence par évaluer une condition. Si c'est vrai, le corps de la boucle est exécuté. Une fois le corps de la boucle exécuté, la condition `While` est à nouveau évaluée pour déterminer s'il convient de réexécuter le corps.

```
Dim iteration As Integer = 1
While iteration <= 10
  Console.Writeline(iteration.ToString() & " ")

  iteration += 1
End While
```

Cela produit:

    1 2 3 4 5 6 7 8 9 10

Attention: Une boucle `While` peut mener à une *boucle infinie* . Considérez ce qui se passerait si la ligne de code qui incrémentait l' `iteration` était supprimée. Dans un tel cas, la condition ne serait jamais vraie et la boucle continuerait indéfiniment.

Do ... Loop

Utilisez `Do...Loop` pour répéter un bloc d'instructions `While` ou `Until` une condition est vraie, en vérifiant la condition au début ou à la fin de la boucle.

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop While x < 10
```

ou

```
Dim x As Integer = 0
Do While x < 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
Loop Until x = 10
```

ou

```
Dim x As Integer = 0
Do Until x = 10
    Console.Write(x & " ")
    x += 1
Loop
```

0 1 2 3 4 5 6 7 8 9

`Continue Do` peut être utilisé pour passer à la prochaine itération de la boucle:

```
Dim x As Integer = 0
Do While x < 10
    x += 1
    If x Mod 2 = 0 Then
        Continue Do
    End If
    Console.Write(x & " ")
Loop
```

1 3 5 7 9

Vous pouvez terminer la boucle avec `Exit Do` - notez que dans cet exemple, l'absence de condition provoquerait une boucle infinie:

```
Dim x As Integer = 0
Do
    Console.Write(x & " ")
    x += 1
    If x = 10 Then
        Exit Do
    End If
Loop
```

0 1 2 3 4 5 6 7 8 9

Court circuit

Toute boucle peut être terminée ou poursuivie à tout moment en utilisant les instructions `Exit` ou `Continue` .

Sortie

Vous pouvez arrêter n'importe quelle boucle en quittant tôt. Pour ce faire, vous pouvez utiliser le mot-clé `Exit` avec le nom de la boucle.

Boucle	Déclaration de sortie
Pour	`Exit For`
Pour chaque	`Exit For`
Faire pendant	`Exit Do`
Tandis que	`Exit While`

Quitter une boucle tôt est un excellent moyen d'améliorer les performances en ne bouclant que le nombre de fois nécessaire pour répondre aux besoins de l'application. Vous trouverez ci-dessous un exemple où la boucle sortira une fois le numéro `2` trouvé.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SoughtValue As Integer = 2
Dim SoughtIndex
For Each i In Numbers
    If i = 2 Then
        SoughtIndex = i
        Exit For
    End If
Next
Debug.Print(SoughtIndex)
```

Continue

Avec la sortie anticipée, vous pouvez également décider de passer à la prochaine itération de la boucle. Cela se fait facilement en utilisant l'instruction `Continue` . Tout comme `Exit` , il est procédé

au nom de la boucle.

Boucle	Continuer déclaration
Pour	`Continue For`
Pour chaque	`Continue For`
Faire pendant	`Continue Do`
Tandis que	`Continue While`

Voici un exemple d'empêcher l'ajout de nombres pairs à la somme.

```
Dim Numbers As Integer() = {1,2,3,4,5}
Dim SumOdd As Integer = 0
For Each i In Numbers
    If Numbers(i) \ 2 = 0 Then Continue For
    SumOdd += Numbers(i)
Next
```

Conseil d'utilisation

Deux techniques alternatives peuvent être utilisées au lieu d'utiliser `Exit` ou `Continue` .

Vous pouvez déclarer une nouvelle variable booléenne en l'initialisant à une valeur et en la définissant conditionnellement sur l'autre valeur dans la boucle. Vous utilisez ensuite une instruction conditionnelle (par exemple `If`) basée sur cette variable pour éviter l'exécution des instructions dans la boucle dans les itérations suivantes.

```
Dim Found As Boolean = False
Dim FoundIndex As Integer
For i As Integer = 0 To N - 1
    If Not Found AndAlso A(i) = SoughtValue Then
        FoundIndex = i
        Found = True
    End If
Next
```

L'une des objections à cette technique est qu'elle peut être inefficace. Par exemple, si dans l'exemple ci-dessus, `N` est 1000000 et que le premier élément du tableau `A` est égal à `SoughtValue` , la boucle `SoughtValue` 999999 autres fois sans rien faire d'utile. Cependant, cette technique peut avoir l'avantage d'une plus grande clarté dans certains cas.

Vous pouvez utiliser l'instruction `GoTo` pour sortir de la boucle. Notez que vous ne pouvez pas utiliser `GoTo` pour sauter *dans* une boucle.

```
    Dim FoundIndex As Integer
    For i As Integer = 0 To N - 1
        If A(i) = SoughtValue Then
            FoundIndex = i
```

```
            GoTo Found
        End If
    Next
    Debug.Print("Not found")
Found:
    Debug.Print(FoundIndex)
```

Cette technique peut parfois être le moyen le plus simple de sortir de la boucle et d'éviter une ou plusieurs instructions exécutées juste après la fin de la boucle.

Vous devez envisager toutes les alternatives et utiliser celle qui correspond le mieux à vos besoins, en tenant compte de l'efficacité, de la rapidité de rédaction du code et de la lisibilité (et donc de la maintenance).

Ne vous laissez pas `GoTo` par `GoTo` dans les cas où il s'agit de la meilleure alternative.

Boucle imbriquée

```
A nested loop is a loop within a loop, an inner loop within the body of an outer one. How this
works is that the first pass of the outer loop triggers the inner loop, which executes to
completion. Then the second pass of the outer loop triggers the inner loop again. This repeats
until the outer loop finishes. a break within either the inner or outer loop would interrupt
this process.
```

La structure d'une boucle imbriquée For Next est:

```
For counter1=startNumber to endNumber (Step increment)

    For counter2=startNumber to endNumber (Step increment)

        One or more VB statements

    Next counter2

Next  counter1
```

Exemple :

```
    For  firstCounter = 1 to 5

      Print "First Loop of " + firstCounter

    For   secondCounter= 1 to 4

      Print "Second Loop of " + secondCounter

    Next secondCounter

  Next firstCounter
```

Lire En boucle en ligne: https://riptutorial.com/fr/vb-net/topic/1639/en-boucle

Chapitre 15: Enum

Examples

Enum définition

Un enum est un ensemble de constantes liées logiquement.

```
Enum Size
    Small
    Medium
    Large
End Enum

Public Sub Order(shirtSize As Size)
    Select Case shirtSize
        Case Size.Small
            ' ...
        Case Size.Medium
            ' ...
        Case Size.Large
            ' ...
    End Select
End Sub
```

Initialisation du membre

Chacun des membres énumérés peut être initialisé avec une valeur. Si une valeur n'est pas spécifiée pour un membre, elle est par défaut initialisée à 0 (s'il s'agit du premier membre de la liste des membres) ou à une valeur supérieure à 1 par rapport à la valeur du membre précédent.

```
Module Module1

    Enum Size
        Small
        Medium = 3
        Large
    End Enum

    Sub Main()
        Console.WriteLine(Size.Small)    ' prints 0
        Console.WriteLine(Size.Medium)   ' prints 3
        Console.WriteLine(Size.Large)    ' prints 4

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

L'attribut Drapeaux

Avec l'attribut `<Flags>` , l'énumération devient un ensemble de drapeaux. Cet attribut permet d'affecter plusieurs valeurs à une variable enum. Les membres d'un enum de drapeaux doivent être initialisés avec des puissances de 2 (1, 2, 4, 8 ...).

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone
        Dim carMaterials as Material = Material.Plastic Or Material.Metal
        Dim knifeMaterials as Material = Material.Metal

        Console.WriteLine(houseMaterials.ToString()) 'Prints "Wood, Stone"
        Console.WriteLine(CType(carMaterials, Integer)) 'Prints 6
    End Sub

End Module
```

HasFlag ()

La méthode `HasFlag()` peut être utilisée pour vérifier si un indicateur est défini.

```
Module Module1

    <Flags>
    Enum Material
        Wood = 1
        Plastic = 2
        Metal = 4
        Stone = 8
    End Enum

    Sub Main()
        Dim houseMaterials As Material = Material.Wood Or Material.Stone

        If houseMaterials.HasFlag(Material.Stone) Then
            Console.WriteLine("the house is made of stone")
        Else
            Console.WriteLine("the house is not made of stone")
        End If
    End Sub

End Module
```

Pour plus d'informations sur l'attribut Flags et son utilisation, consultez la documentation officielle de Microsoft .

Analyse de chaîne

Une instance Enum peut être créée en analysant une représentation sous forme de chaîne de Enum.

```
Module Module1

    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size =  DirectCast([Enum].Parse(GetType(Size), "Medium"), Size)

        ' Prints 'Medium'
        Console.WriteLine(shirtSize.ToString())

        ' Waits until user presses any key
        Console.ReadKey()
    End Sub

End Module
```

Voir aussi: Analyse une chaîne en une valeur Enum dans VB.NET

GetNames ()

Retourne les noms des constantes dans Enum spécifié en tant que tableau de chaînes:

```
Module Module1

  Enum Size
    Small
    Medium
    Large
  End Enum

  Sub Main()
    Dim sizes = [Enum].GetNames(GetType(Size))

    For Each size In sizes
      Console.WriteLine(size)
      Next
  End Sub

End Module
```

Sortie:

Petit

Moyen

Grand

GetValues ()

'Cette méthode est utile pour itérer les valeurs Enum'

```
Enum Animal
    Dog = 1
    Cat = 2
    Frog = 4
End Enum

Dim Animals = [Enum].GetValues(GetType(Animal))

For Each animal in Animals
    Console.WriteLine(animal)
Next
```

Impressions:

1

2

4

ToString ()

La méthode `ToString` sur une énumération renvoie le nom de la chaîne de l'énumération. Par exemple:

```
Module Module1
    Enum Size
        Small
        Medium
        Large
    End Enum

    Sub Main()
        Dim shirtSize As Size = Size.Medium
        Dim output As String = shirtSize.ToString()
        Console.WriteLine(output)  ' Writes "Medium"
    End Sub
End Module
```

Si, toutefois, la représentation sous forme de chaîne de la valeur entière réelle de l'enum est souhaitée, vous pouvez convertir l'énumération en un `Integer` puis appeler `ToString` :

```
Dim shirtSize As Size = Size.Medium
Dim output As String = CInt(shirtSize).ToString()
Console.WriteLine(output)  ' Writes "1"
```

Déterminer si un Enum a FlagsAttribute spécifié ou non

L'exemple suivant peut être utilisé pour déterminer si une énumération a le **paramètre**

FlagsAttribute spécifié. La méthodologie utilisée est basée sur la **réflexion** .

Cet exemple donnera un résultat `True` :

```
Dim enu As [Enum] = New FileAttributes()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Cet exemple donnera un résultat `False` :

```
Dim enu As [Enum] = New ConsoleColor()
Dim hasFlags As Boolean = enu.GetType().GetCustomAttributes(GetType(FlagsAttribute),
inherit:=False).Any()
Console.WriteLine("{0} Enum has FlagsAttribute?: {1}", enu.GetType().Name, hasFlags)
```

Nous pouvons concevoir une méthode d'extension générique comme celle-ci:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function HasFlagsAttribute(ByVal sender As [Enum]) As Boolean
    Return sender.GetType().GetCustomAttributes(GetType(FlagsAttribute), inherit:=False).Any()
End Function
```

Exemple d'utilisation:

```
Dim result As Boolean = (New FileAttributes).HasFlagsAttribute()
```

Indicateur For-each (itération de drapeau)

Dans certains scénarios très spécifiques, nous ressentirions le besoin d'effectuer une action spécifique pour chaque indicateur de l'énumération source.

Nous pouvons écrire une méthode simple d'extension *générique* pour réaliser cette tâche.

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Sub ForEachFlag(Of T)(ByVal sender As [Enum],
                             ByVal action As Action(Of T))

    For Each flag As T In sender.Flags(Of T)
        action.Invoke(flag)
    Next flag

End Sub
```

Exemple d'utilisation:

```
Dim flags As FileAttributes = (FileAttributes.ReadOnly Or FileAttributes.Hidden)
```

```
flags.ForEachFlag(Of FileAttributes)(
      Sub(ByVal x As FileAttributes)
          Console.WriteLine(x.ToString())
      End Sub)
```

Déterminer la quantité de drapeaux dans une combinaison de drapeaux

L'exemple suivant est destiné à compter la quantité d'indicateurs dans la combinaison d'indicateurs spécifiée.

L'exemple est fourni en tant que méthode d'extension:

```
<DebuggerStepThrough>
<Extension>
<EditorBrowsable(EditorBrowsableState.Always)>
Public Function CountFlags(ByVal sender As [Enum]) As Integer
    Return sender.ToString().Split(","c).Count()
End Function
```

Exemple d'utilisation:

```
Dim flags As FileAttributes = (FileAttributes.Archive Or FileAttributes.Compressed)
Dim count As Integer = flags.CountFlags()
Console.WriteLine(count)
```

Trouver la valeur la plus proche dans un Enum

Le code suivant illustre comment trouver la valeur la plus proche d'un **Enum** .

Nous définissons d'abord ce **Enum** qui servira à spécifier des critères de recherche (direction de recherche)

```
Public Enum EnumFindDirection As Integer
    Nearest = 0
    Less = 1
    LessOrEqual = 2
    Greater = 3
    GreaterOrEqual = 4
End Enum
```

Et maintenant, nous implémentons l'algorithme de recherche:

```
<DebuggerStepThrough>
Public Shared Function FindNearestEnumValue(Of T)(ByVal value As Long,
                                                  ByVal direction As EnumFindDirection) As T

    Select Case direction

        Case EnumFindDirection.Nearest
            Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                    Order By Math.Abs(value - Convert.ToInt64(enumValue))
                    ).FirstOrDefault
```

```
        Case EnumFindDirection.Less
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) < value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.LessOrEqual
            If value < Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).First) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).FirstOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) <= value
                        ).LastOrDefault
            End If

        Case EnumFindDirection.Greater
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) > value
                        ).FirstOrDefault
            End If

        Case EnumFindDirection.GreaterOrEqual
            If value > Convert.ToInt64([Enum].GetValues(GetType(T)).Cast(Of T).Last) Then
                Return [Enum].GetValues(GetType(T)).Cast(Of T).LastOrDefault

            Else
                Return (From enumValue As T In [Enum].GetValues(GetType(T)).Cast(Of T)()
                        Where Convert.ToInt64(enumValue) >= value
                        ).FirstOrDefault
            End If

    End Select

End Function
```

Exemple d'utilisation:

```
Public Enum Bitrate As Integer
    Kbps128 = 128
    Kbps192 = 192
    Kbps256 = 256
    Kbps320 = 320
End Enum

Dim nearestValue As Bitrate = FindNearestEnumValue(Of Bitrate)(224,
EnumFindDirection.GreaterOrEqual)
```

Lire Enum en ligne: https://riptutorial.com/fr/vb-net/topic/1809/enum

Chapitre 16: Filetage

Examples

Effectuer des appels thread-safe à l'aide de Control.Invoke ()

À l'aide de la méthode `Control.Invoke()` , vous pouvez déplacer l'exécution d'une méthode ou d'une fonction d'un thread d'arrière-plan vers le thread sur lequel le contrôle a été créé, à savoir le thread d'interface utilisateur. Ce faisant, votre code sera mis en file d'attente pour s'exécuter sur le thread du contrôle à la place, ce qui supprime la possibilité de simultanéité.

La propriété `Control.InvokeRequired` doit également être vérifiée afin de déterminer si vous devez appeler ou si le code s'exécute déjà sur le même thread que le contrôle.

La méthode `Invoke()` prend un délégué comme premier paramètre. Un délégué détient la référence, la liste de paramètres et le type de retour dans une autre méthode.

Dans Visual Basic 2010 (10.0) ou supérieur, *les expressions lambda* peuvent être utilisées pour créer une méthode de délégation à la volée:

```
If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(Sub() LogTextBox.AppendText("Check passed"))
Else
    LogTextBox.AppendText("Check passed")
End If
```

Considérant que dans Visual Basic 2008 (9.0) ou inférieur, vous devez déclarer le délégué vous-même:

```
Delegate Sub AddLogText(ByVal Text As String)

If LogTextBox.InvokeRequired = True Then
    LogTextBox.Invoke(New AddLogText(AddressOf UpdateLog), "Check passed")
Else
    UpdateLog("Check passed")
End If

Sub UpdateLog(ByVal Text As String)
    LogTextBox.AppendText(Text)
End Sub
```

Effectuer des appels sécurisés avec Async / Await

Si nous essayons de changer un objet sur le thread d'interface utilisateur à partir d'un thread différent, nous aurons une exception d'opération croisée:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' Cross thread-operation exception as the assignment is executed on a different thread
    ' from the UI one:
```

```
    Task.Run(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Avant **VB 14.0** et **.NET 4.5,** la solution invoquait l'affectation et l'objet vivant sur le thread d'interface utilisateur:

```
Private Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the conde on the UI thread:
    MyButton.Invoke(Sub() MyButton.Text = Thread.CurrentThread.ManagedThreadId)
End Sub
```

Avec **VB 14.0** , nous pouvons exécuter une `Task` sur un thread différent, puis restaurer le contexte une fois l'exécution terminée, puis effectuer l'assignation avec Async / Await:

```
Private Async Sub Button_Click(sender As Object, e As EventArgs) Handles MyButton.Click
    ' This will run the code on a different thread then the context is restored
    ' so the assignment happens on the UI thread:
    MyButton.Text = Await Task.Run(Function() Thread.CurrentThread.ManagedThreadId)
End Sub
```

Lire Filetage en ligne: https://riptutorial.com/fr/vb-net/topic/1913/filetage

Chapitre 17: Fonctionnalités de Visual Basic 14.0

Introduction

Visual Basic 14 est la version de Visual Basic fournie avec Visual Studio 2015.

Cette version a été réécrite à partir de zéro dans environ 1,3 million de lignes de VB. De nombreuses fonctionnalités ont été ajoutées pour supprimer les irritations les plus courantes et rendre les modèles de codage courants plus propres.

Le numéro de version de Visual Basic est passé directement de 12 à 14, en sautant 13. Cela a été fait pour garder VB en ligne avec la numérotation de la version de Visual Studio elle-même.

Examples

Opérateur conditionnel nul

Pour éviter la vérification nulle verbeuse, le `?.` l'opérateur a été introduit dans la langue.

L'ancienne syntaxe verbeuse:

```
If myObject IsNot Nothing AndAlso myObject.Value >= 10 Then
```

Peut être maintenant remplacé par le concis:

```
If myObject?.Value >= 10 Then
```

Le `?` l'opérateur est particulièrement puissant lorsque vous avez une chaîne de propriétés. Considérer ce qui suit:

```
Dim fooInstance As Foo = Nothing
Dim s As String
```

Normalement, vous devriez écrire quelque chose comme ceci:

```
If fooInstance IsNot Nothing AndAlso fooInstance.BarInstance IsNot Nothing Then
    s = fooInstance.BarInstance.Baz
Else
    s = Nothing
End If
```

Mais avec le `?` l'opérateur cela peut être remplacé par seulement:

```
s = fooInstance?.BarInstance?.Baz
```

Opérateur NameOf

L'opérateur `NameOf` résout les espaces de noms, les types, les variables et les noms de membres au moment de la compilation et les remplace par l'équivalent de chaîne.

Un des cas d'utilisation:

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException("variable")
End Sub
```

L'ancienne syntaxe exposera le risque de renommer la variable et de laisser la chaîne codée en dur à la mauvaise valeur.

```
Sub MySub(variable As String)
    If variable Is Nothing Then Throw New ArgumentNullException(NameOf(variable))
End Sub
```

Avec `NameOf` , renommer la variable uniquement provoquera une erreur de compilation. Cela permettra également à l'outil de renommage de renommer les deux en un seul effort.

L'opérateur `NameOf` n'utilise que le dernier composant de la référence entre crochets. Ceci est important lorsque vous manipulez quelque chose comme des espaces de noms dans l'opérateur `NameOf` .

```
Imports System

Module Module1
    Sub WriteIO()
        Console.WriteLine(NameOf(IO)) 'displays "IO"
        Console.WriteLine(NameOf(System.IO)) 'displays "IO"
    End Sub
End Module
```

L'opérateur utilise également le nom de la référence saisie sans résoudre aucun changement de nom. Par exemple:

```
Imports OldList = System.Collections.ArrayList

Module Module1
    Sub WriteList()
        Console.WriteLine(NameOf(OldList)) 'displays "OldList"
        Console.WriteLine(NameOf(System.Collections.ArrayList)) 'displays "ArrayList"
    End Sub
End Module
```

Interpolation de chaîne

Cette nouvelle fonctionnalité rend la concaténation de chaîne plus lisible. Cette syntaxe sera

compilée à son appel `String.Format` équivalent.

Sans interpolation de chaîne:

```
String.Format("Hello, {0}", name)
```

Avec interpolation de chaîne:

```
$"Hello, {name}"
```

Les deux lignes sont équivalentes et les deux sont compilées en un appel à `String.Format` .

Comme dans `String.Format` , les parenthèses peuvent contenir n'importe quelle expression unique (appel à une méthode, propriété, opérateur de coalescence nul et ainsi de suite).

L'interpolation de chaîne est la méthode préférée par rapport à `String.Format` car elle empêche certaines erreurs d'exécution de se produire. Considérons la ligne `String.Format` suivante:

```
String.Format("The number of people is {0}/{1}", numPeople)
```

Cela compilera, mais provoquera une erreur d'exécution car le compilateur ne vérifie pas que le nombre d'arguments correspond aux espaces réservés.

Propriétés automatiques en lecture seule

Les propriétés en lecture seule étaient toujours possibles dans VB.NET dans ce format:

```
Public Class Foo

  Private _MyProperty As String = "Bar"

  Public ReadOnly Property MyProperty As String
    Get
      Return _MyProperty
    End Get
  End Property

End Class
```

La nouvelle version de Visual Basic permet une courte main pour la déclaration de propriété comme suit:

```
Public Class Foo

  Public ReadOnly Property MyProperty As String = "Bar"

End Class
```

L'implémentation réelle générée par le compilateur est exactement la même pour les deux exemples. La nouvelle méthode pour l'écrire n'est qu'une main courte. Le compilateur générera toujours un champ privé au format: `_<PropertyName>` pour sauvegarder la propriété en lecture seule.

Modules partiels et interfaces

Semblable à des classes partielles, la nouvelle version de Visual Basic est désormais capable de gérer des modules partiels et des interfaces partielles. La syntaxe et le comportement sont exactement les mêmes que pour les classes partielles.

Un exemple de module partiel:

```
Partial Module Module1
  Sub Main()
    Console.Write("Ping -> ")
    TestFunktion()
  End Sub
End Module

Partial Module Module1
  Private Sub TestFunktion()
    Console.WriteLine("Pong")
  End Sub
End Module
```

Et une interface partielle:

```
Partial Interface Interface1
  Sub Methode1()
End Interface

Partial Interface Interface1
  Sub Methode2()
End Interface

Public Class Class1
  Implements Interface1
  Public Sub Methode1() Implements Interface1.Methode1
    Throw New NotImplementedException()
  End Sub

  Public Sub Methode2() Implements Interface1.Methode2
    Throw New NotImplementedException()
  End Sub
End Class
```

Tout comme pour les classes partielles, les définitions des modules partiels et des interfaces doivent être situées dans le même espace de noms et le même assemblage. Cela est dû au fait que les parties partielles des modules et des interfaces sont fusionnées pendant la compilation et que l'assembly compilé ne contient aucune indication indiquant que la définition d'origine du module ou de l'interface était fractionnée.

Littéraux multilignes

VB autorise désormais les littéraux de chaîne répartis sur plusieurs lignes.

Ancienne syntaxe:

```
Dim text As String = "Line1" & Environment.NewLine & "Line2"
```

Nouvelle syntaxe:

```
Dim text As String = "Line 1
Line 2"
```

Amélioration de la directive #Region

La directive #Region peut maintenant être placée dans des méthodes et peut même couvrir des méthodes, des classes et des modules.

```
#Region "A Region Spanning A Class and Ending Inside Of A Method In A Module"
   Public Class FakeClass
    'Nothing to see here, just a fake class.
   End Class

   Module Extensions

    ''' <summary>
    ''' Checks the path of files or directories and returns [TRUE] if it exists.
    ''' </summary>
    ''' <param name="Path">[Sting] Path of file or directory to check.</param>
    ''' <returns>[Boolean]</returns>
    <Extension>
    Public Function PathExists(ByVal Path As String) As Boolean
        If My.Computer.FileSystem.FileExists(Path) Then Return True
        If My.Computer.FileSystem.DirectoryExists(Path) Then Return True
        Return False
    End Function

    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
#End Region
        Return Split(Split(Assy.FullName, ",")(1), "=")(1)
    End Function
End Module
```

Commentaires après suite de ligne implicite

VB 14.0 introduit la possibilité d'ajouter des commentaires après la continuation implicite des lignes.

```
Dim number =
    From c As Char 'Comment
    In "dj58kwd92n4" 'Comment
    Where Char.IsNumber(c) 'Comment
    Select c 'Comment
```

Gestion des exceptions

Pendant le codage, des erreurs inattendues surviennent assez fréquemment, ce qui nécessite un débogage et des tests. Mais parfois les erreurs sont effectivement attendues et pour le contourner, il y a le bloc `Try..Catch..Throw..Finally..End Try` .

Pour gérer correctement une erreur, le code est placé dans un bloc `Try..Catch` , dans lequel le `Catch` , comme le nom l'indique, intercepte toutes les exceptions qui surviennent dans ce bloc.

Et en cas d'exception, nous avons la possibilité de `Throw` l'erreur, c'est-à-dire de la renvoyer pour avertir l'utilisateur ou la gérer en interne dans le code lui-même.

La partie `Finally` est le code final qui, quel que soit le résultat, s'il y a une exception ou non, le code s'exécutera avant de sortir du bloc.

Si vous avez besoin de sortir de l'horloge, vous pouvez utiliser l'instruction `Exit Try` . Mais ici aussi, le code de la section `Finally` sera exécuté avant de se terminer.

La syntaxe est simple

```
Try
    [ tryStatements ]
    [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
    [ catchStatements ]
    [ Exit Try ] ]
[ Catch ... ]
[ Finally
    [ finallyStatements ] ]
End Try
```

où seul le `Try` et `End Try` est obligatoire. Le reste peut être ignoré mais, en tant que bonne pratique, incluez la partie `Finally` , même si elle reste vide.

À l’exception, il existe différents types d’exceptions pouvant être interceptées. Ce sont des exceptions prêtes à l'emploi disponibles dans le cadre .Net, comme ci-dessous;

Classe d'exception	Brève description
System.IO.IOException	Gère les erreurs d'E / S
System.IndexOutOfRangeException	Fait référence à un index de tableau hors limites
System.ArrayTypeMismatchException	Lorsque le type ne correspond pas au type de tableau
System.NullReferenceException	Gère les erreurs générées lors du référencement d'un objet nul.
System.DivideByZeroException	Gère les erreurs générées par la division d'un dividende avec zéro.

Classe d'exception	Brève description
System.InvalidCastException	Gère les erreurs générées lors de la conversion.
System.OutOfMemoryException	Gère les erreurs générées par une mémoire libre insuffisante.
System.StackOverflowException	Gère les erreurs générées par le débordement de pile.
----------	----------

Lire Fonctionnalités de Visual Basic 14.0 en ligne: https://riptutorial.com/fr/vb-net/topic/1501/fonctionnalites-de-visual-basic-14-0

Chapitre 18: GDI +

Examples

Créer un objet graphique

Il existe trois façons de créer un objet graphique

1. De l' **événement de peinture**

Chaque fois que le contrôle est redessiné (redimensionné, actualisé ...), cet événement est appelé. Utilisez cette méthode si vous souhaitez que le contrôle dessine de manière cohérente sur le contrôle.

```
'this will work on any object's paint event, not just the form
Private Sub Form1_Paint(sender as Object, e as PaintEventArgs) Handles Me.Paint
  Dim gra as Graphics
  gra = e.Graphics
End Sub
```

2. **Créer un graphique**

Ceci est le plus souvent utilisé lorsque vous souhaitez créer un graphique unique sur le contrôle, ou vous ne voulez pas que le contrôle se repeigne

```
Dim btn as New Button
Dim g As Graphics = btn.CreateGraphics
```

3. À partir d'un **graphique existant**

Utilisez cette méthode pour dessiner et modifier un graphique existant

```
'The existing image can be from a filename, stream or Drawing.Graphic
Dim image = New Bitmap("C:\TempBit.bmp")
Dim gr As Graphics = Graphics.FromImage(image)
```

Dessiner des formes

Pour commencer à dessiner une forme, vous devez définir un objet stylo. Le `Pen` accepte deux paramètres:

1. Couleur du stylo ou pinceau
2. Largeur du stylo

L'objet Plume est utilisé pour créer un **contour** de l'objet que vous souhaitez dessiner

Après avoir défini le stylet, vous pouvez définir des propriétés de stylet spécifiques

```
   Dim pens As New Pen(Color.Purple)
   pens.DashStyle = DashStyle.Dash 'pen will draw with a dashed line
   pens.EndCap = LineCap.ArrowAnchor 'the line will end in an arrow
   pens.StartCap = LineCap.Round 'The line draw will start rounded
   '*Notice* - the Start and End Caps will not show if you draw a closed shape
```

Ensuite, utilisez l'objet graphique que vous avez créé pour dessiner la forme

```
  Private Sub GraphicForm_Paint(sender As Object, e As PaintEventArgs) Handles MyBase.Paint
    Dim pen As New Pen(Color.Blue, 15) 'Use a blue pen with a width of 15
    Dim point1 As New Point(5, 15) 'starting point of the line
    Dim point2 As New Point(30, 100) 'ending point of the line
    e.Graphics.DrawLine(pen, point1, point2)

    e.Graphics.DrawRectangle(pen, 60, 90, 200, 300) 'draw an outline of the rectangle
```

Par défaut, la largeur du stylet est égale à 1

```
    Dim pen2 as New Pen(Color.Orange) 'Use an orange pen with width of 1
    Dim origRect As New Rectangle(90, 30, 50, 60) 'Define bounds of arc
    e.Graphics.DrawArc(pen2, origRect, 20, 180) 'Draw arc in the rectangle bounds

End Sub
```

Remplir des formes

Graphics.FillShapes dessine une forme et la remplit avec la couleur donnée. Les formes de remplissage peuvent utiliser

1. Outil `Brush` - pour remplir une forme avec une couleur unie

   ```
   Dim rect As New Rectangle(50, 50, 50, 50)
   e.Graphics.FillRectangle(Brushes.Green, rect) 'draws a rectangle that is filled with
   green

   e.Graphics.FillPie(Brushes.Silver, rect, 0, 180) 'draws a half circle that is filled with
   silver
   ```

2. `HatchBrush` Tool - pour remplir la forme avec un motif

```
Dim hBrush As New HatchBrush(HatchStyle.ZigZag, Color.SkyBlue, Color.Gray)
'creates a HatchBrush Tool with a background color of blue, foreground color of gray,
'and will fill with a zigzag pattern
Dim rectan As New Rectangle(100, 100, 100, 100)
e.Graphics.FillRectangle(hBrush, rectan)
```

3. `LinearGradientBrush` - pour remplir une forme avec un dégradé

   ```
   Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed,
   Color.PaleGreen)
    Dim rect As New Rectangle(50, 50, 200, 200)
    e.Graphics.FillRectangle(lBrush, rect)
   ```

4. `TextureBrush` - pour remplir la forme avec une image

Vous pouvez choisir une image parmi les ressources, une bitmap déjà définie ou un nom de fichier

```
   Dim textBrush As New TextureBrush(New Bitmap("C:\ColorPic.jpg"))
    Dim rect As New Rectangle(400, 400, 100, 100)
    e.Graphics.FillPie(textBrush, rect, 0, 360)
```

L' `Hatch Brush Tool` et `LinearGradientBrush` importent tous deux l'instruction suivante: **Imports System.Drawing.Drawing2D**

Texte

Pour dessiner du texte sur le formulaire, utilisez la méthode `DrawString`

Lorsque vous dessinez une chaîne, vous pouvez utiliser l'une des 4 brosses répertoriées ci-dessus.

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
e.Graphics.DrawString("HELLO", New Font("Impact", 60, FontStyle.Bold), lBrush, New Point(40,
400))
'this will draw the word "Hello" at the given point, with a linearGradient Brush
```

Comme vous ne pouvez pas définir la largeur ou la hauteur du texte, utilisez l' `Measure Text` pour vérifier la taille du texte.

```
Dim lBrush As New LinearGradientBrush(point1, point2, Color.MediumVioletRed, Color.PaleGreen)
Dim TextSize =  e.Graphics.MeasureString("HELLO", New Font("Impact", 60, FontStyle.Bold),
lBrush)
'Use the TextSize to determine where to place the string, or if the font needs to be smaller
```

> Ex: Vous devez dessiner le mot "Test" en haut du formulaire. La largeur du formulaire est de 120. Utilisez cette boucle pour réduire la taille de la police jusqu'à ce qu'elle rentre dans la largeur des formulaires.

```
Dim FontSize as Integer = 80
Dim TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Do while TextSize.Width >120
FontSize = FontSize -1
TextSize = e.graphics.measeString("Test", New Font("Impact",FontSize, FontStyle.Bold), new
Brush(colors.Blue, 10)
Loop
```

Lire GDI + en ligne: https://riptutorial.com/fr/vb-net/topic/5096/gdi-plus

Chapitre 19: Génériques

Examples

Créez une classe générique

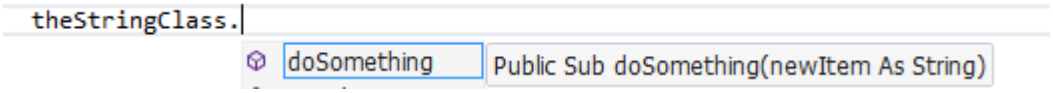
Un type générique est créé pour s'adapter afin que la même fonctionnalité puisse être accessible pour différents types de données.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Instance d'une classe générique

En créant une instance de la même classe avec un type différent, l'interface de la classe change en fonction du type donné.

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



Définir une classe 'générique'

Une classe générique est une classe qui s’adapte à un type donné ultérieurement afin que la même fonctionnalité puisse être proposée à différents types.

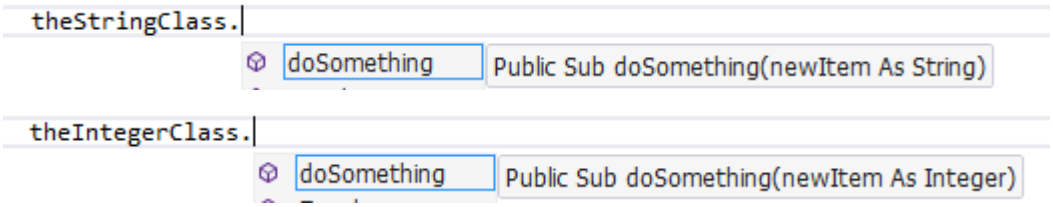
Dans cet exemple de base, une classe générique est créée. Il a un sous qui utilise le type générique T. En programmant cette classe, nous ne connaissons pas le type de T. Dans ce cas, T a toutes les caractéristiques de Object.

```
Public Class SomeClass(Of T)
    Public Sub doSomething(newItem As T)
        Dim tempItem As T
        ' Insert code that processes an item of data type t.
    End Sub
End Class
```

Utiliser une classe générique

Dans cet exemple, 2 instances de la classe SomeClass ont été créées. Selon le type donné, les 2 instances ont une interface différente:

```
Dim theStringClass As New SomeClass(Of String)
Dim theIntegerClass As New SomeClass(Of Integer)
```



La classe générique la plus connue est List (of)

Limiter les types possibles donnés

Les types possibles transmis à une nouvelle instance de SomeClass doivent hériter SomeBaseClass. Cela peut aussi être une interface. Les caractéristiques de SomeBaseClass sont accessibles dans cette définition de classe.

```
Public Class SomeClass(Of T As SomeBaseClass)
    Public Sub DoSomething(newItem As T)
        newItem.DoSomethingElse()
        ' Insert code that processes an item of data type t.
    End Sub
End Class

Public Class SomeBaseClass
    Public Sub DoSomethingElse()
    End Sub
End Class
```

Créer une nouvelle instance du type donné

La création d'une nouvelle intent d'un type générique peut être faite / vérifiée au moment de la compilation.

```
Public Class SomeClass(Of T As {New})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class
```

Ou avec des types limités:

```
Public Class SomeClass(Of T As {New, SomeBaseClass})
    Public Function GetInstance() As T
        Return New T
    End Function
End Class

Public Class SomeBaseClass
End Class
```

La baseClass (si l'on ne lui donne aucun objet) doit avoir un paramètre constructeur moins.

Cela peut également être fait à l'exécution par réflexion

Lire Génériques en ligne: https://riptutorial.com/fr/vb-net/topic/6572/generiques

Chapitre 20: Google Maps dans un formulaire Windows

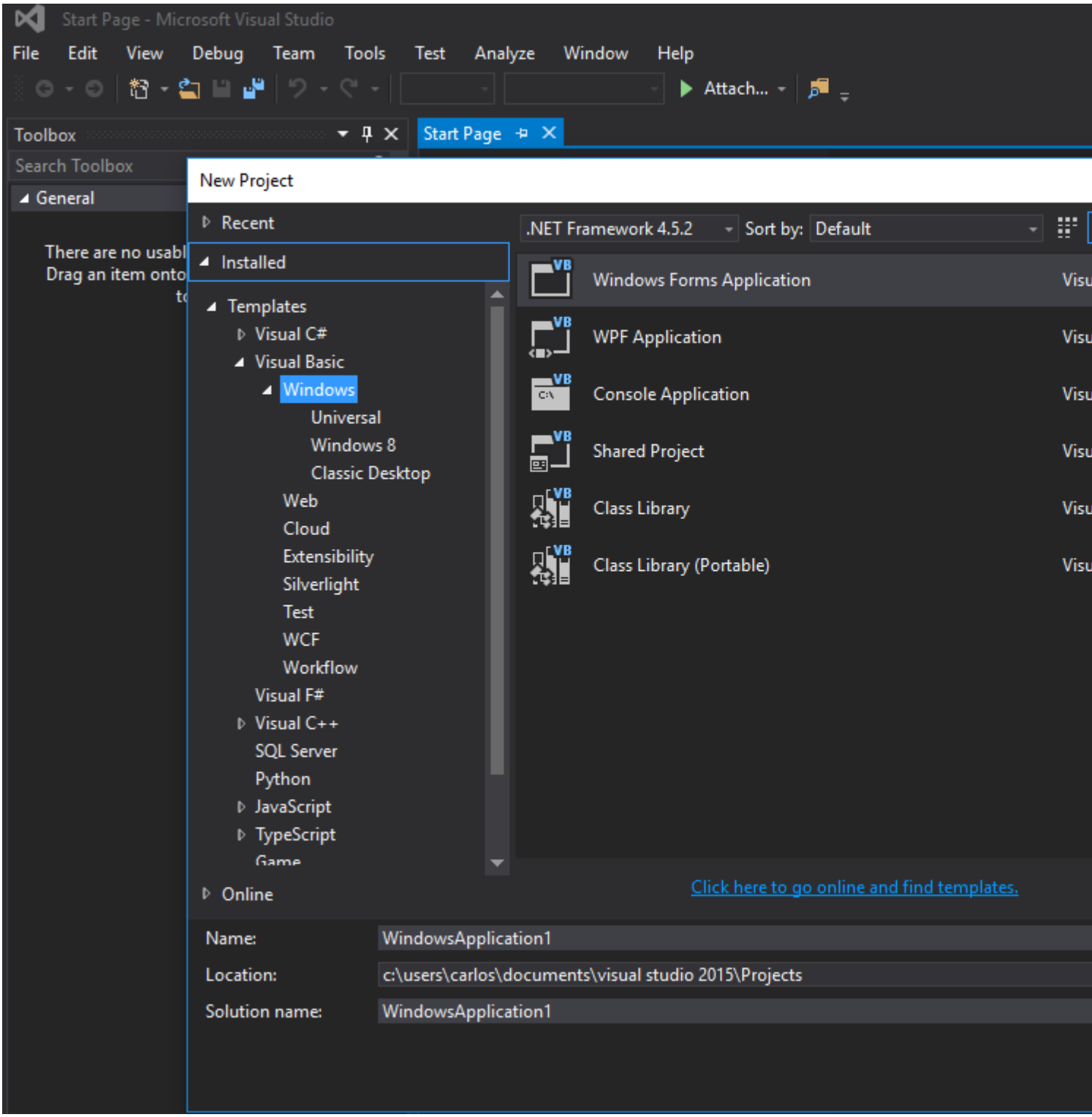
Examples

Comment utiliser une carte Google dans un Windows Form

La première partie de cet exemple explique comment le mettre en œuvre. Dans la seconde, je vais vous expliquer comment cela fonctionne. Cela essaie d'être un exemple général. Le modèle pour la carte (voir l'étape 3) et les fonctions d'exemple sont entièrement personnalisables.

########## **LA MISE EN OEUVRE** ##########
##########

Étape 1. Commencez par créer un nouveau projet et sélectionnez Application Windows Form. Laissons son nom comme "Form1".



Étape 2. Ajoutez un contrôle WebBrowser (qui contiendra votre carte) à votre Form1. Appelons ça "wbmap"

Étape 3. Créez un fichier .html nommé "googlemap_template.html" avec votre éditeur de texte préféré et collez le code suivant:

googlemap_template.html

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
     <style type="text/css">
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #gmap {
        height: 100%;
      }
     </style>
    <script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript">
        function initialize() {
            //Use window.X instead of var X to make a variable globally available
            window.markers = new Array();
            window.marker_data = [[MARKER_DATA]];
            window.gmap = new google.maps.Map(document.getElementById('gmap'), {
            zoom: 15,
            center: new google.maps.LatLng(marker_data[0][0], marker_data[0][1]),
            mapTypeId: google.maps.MapTypeId.ROADMAP
          });
          var infowindow = new google.maps.InfoWindow();
          var newmarker, i;
          for (i = 0; i < marker_data.length; i++) {
              if (marker_data[0].length == 2) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap
                  });
              } else if (marker_data[0].length == 3) {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2])
                  });
              } else {
                  newmarker = new google.maps.Marker({
                      position: new google.maps.LatLng(marker_data[i][0], marker_data[i][1]),
                      map: gmap,
                      title: (marker_data[i][2]),
                      icon: (marker_data[i][3])
                  });
              }
            google.maps.event.addListener(newmarker, 'click', (function (newmarker, i) {
                return function () {
                    if (newmarker.title) {
                        infowindow.setContent(newmarker.title);
                        infowindow.open(gmap, newmarker);
                    }
                    gmap.setCenter(newmarker.getPosition());
                    // Calling functions written in the WF
                    window.external.showVbHelloWorld();
                    window.external.getMarkerDataFromJavascript(newmarker.title,i);
                }
            })(newmarker, i));
            markers[i] = newmarker;
          }
        }
```

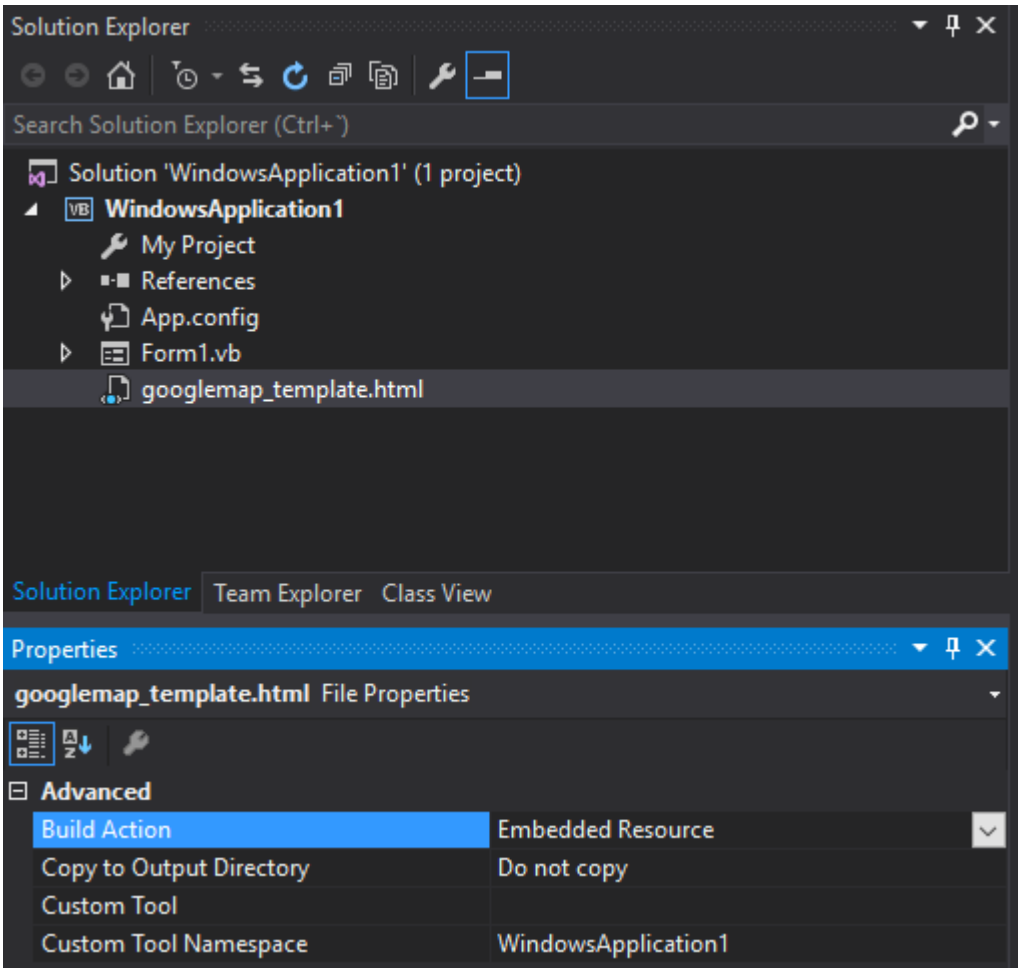
```
        google.maps.event.addDomListener(window, 'load', initialize);
    </script>
    <script type="text/javascript">
        // Function triggered from the WF with no arguments
        function showJavascriptHelloWorld() {
            alert("Hello world in HTML from WF");
        }
     </script>
      <script type="text/javascript">
        // Function triggered from the WF with a String argument
        function focusMarkerFromIdx(idx) {
            google.maps.event.trigger(markers[idx], 'click');
        }
      </script>
  </head>
  <body>
    <div id="gmap"></div>
  </body>
</html>
```

Cela servira de notre modèle de carte. Je vais vous expliquer comment cela fonctionne plus tard.

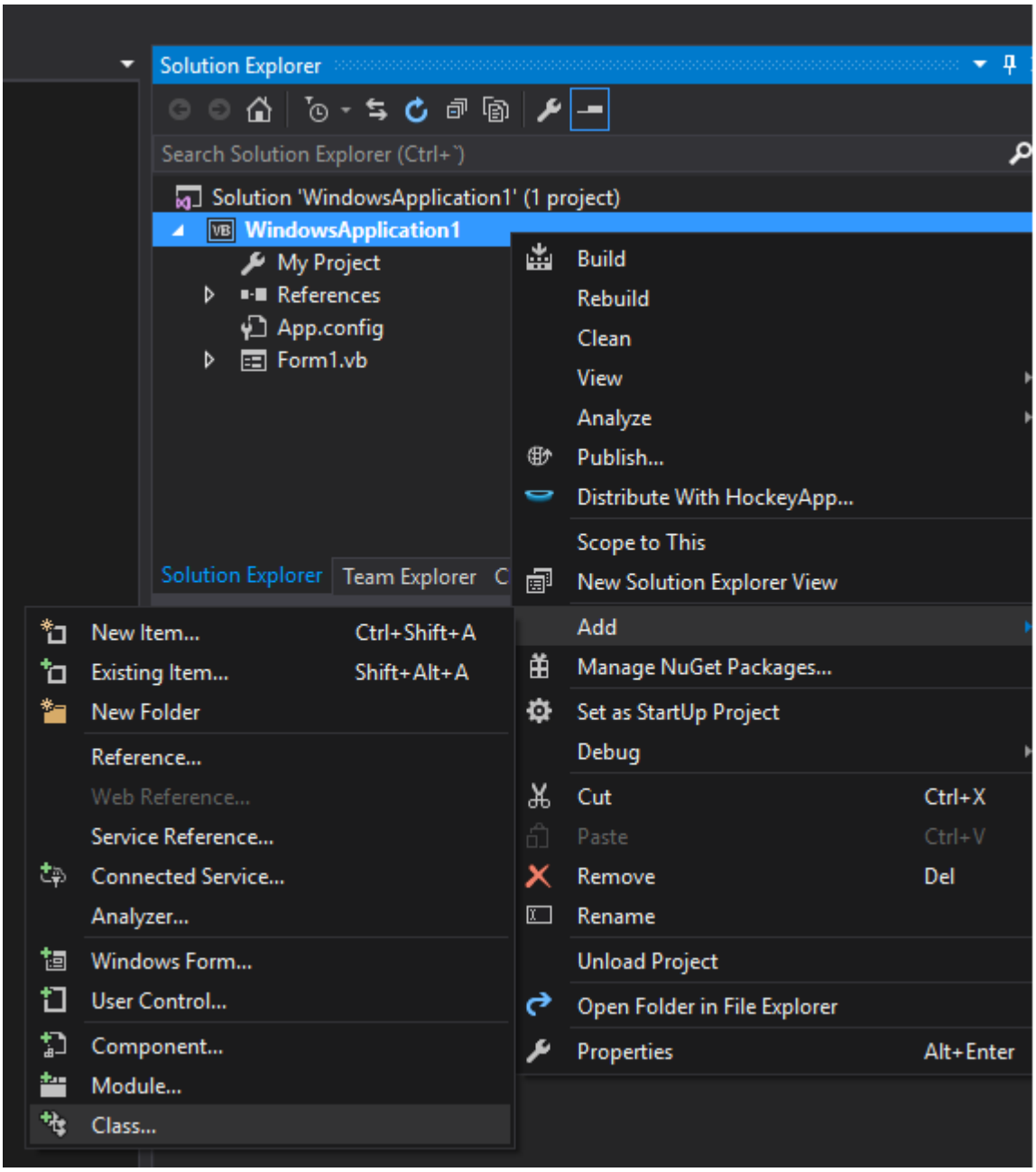
Étape 4. Ajoutez le fichier googlemap_template.hmtl à votre projet (clic droit sur votre projet-> add-> élément existant)

Étape 5. Une fois qu'il apparaît dans votre Explorateur de solutions, définissez ses propriétés sur:

- Build Action -> Ressource intégrée
- Espace de noms de l'outil personnalisé -> écrivez le nom du projet



Étape 6. Ajouter une nouvelle classe (clic droit sur votre projet-> add-> class). Dans mon exemple, je l'appellerai GoogleMapHelper.



Étape 7. Collez le code suivant dans votre classe:

GoogleMapHelper.vb

```
    Imports System.IO
    Imports System.Reflection
    Imports System.Text

    Public Class GoogleMapHelper

    ' 1- googlemap_template.html must be copied in the main project folder
    ' 2- add the file into the Visual Studio Solution Explorer (add existing file)
    ' 3- set the properties of the file to:
    '                                  Build Action -> Embedded Resource
    '                                  Custom Tool Namespace -> write the name of the project

    Private Const ICON_FOLDER As String = "marker_icons/" 'images must be stored in a folder
inside  Debug/Release folder
    Private Const MAP_TEMPLATE As String = "WindowsApplication1.googlemap_template.html"
```

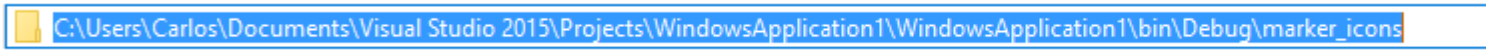
```
    Private Const TEXT_TO_REPLACE_MARKER_DATA As String = "[[MARKER_DATA]]"
    Private Const TMP_NAME As String = "tmp_map.html"


    Private mWebBrowser As WebBrowser

    'MARKER POSITIONS
    Private mPositions As Double(,) 'lat, lon
    ' marker data allows different formats to include lat,long and optionally title and icon:
    ' op1: mMarkerData = New String(N-1, 1) {{lat1, lon1}, {lat2, lon2}, {latN, lonN}}
    ' op2: mMarkerData = New String(N-1, 2) {{lat1, lon1,'title1'}, {lat2, lon2,'title2'},
{latN, lonN, 'titleN'}}
    ' op3: mMarkerData = New String(N-1, 3) {{lat1, lon1,'title1','image1.png'}, {lat2,
lon2,'title2','image2.png'}, {latN, lonN, 'titleN','imageN.png'}}
    Private mMarkerData As String(,) = Nothing


    Public Sub New(ByRef wb As WebBrowser, pos As Double(,))
        mWebBrowser = wb
        mPositions = pos
        mMarkerData = getMarkerDataFromPositions(pos)
    End Sub

    Public Sub New(ByRef wb As WebBrowser, md As String(,))
        mWebBrowser = wb
        mMarkerData = md
    End Sub

    Public Sub loadMap()
        mWebBrowser.Navigate(getMapTemplate())
    End Sub

    Private Function getMapTemplate() As String

        If mMarkerData Is Nothing Or mMarkerData.GetLength(1) > 4 Then
            MessageBox.Show("Marker data has not the proper size. It must have 2, 3 o 4
columns")
            Return Nothing
        End If

        Dim htmlTemplate As New StringBuilder()
        Dim tmpFolder As String = Environment.GetEnvironmentVariable("TEMP")
        Dim dataSize As Integer = mMarkerData.GetLength(1) 'number of columns
        Dim mMarkerDataAsText As String = String.Empty
        Dim myresourcePath As String = My.Resources.ResourceManager.BaseName
        Dim myresourcefullPath As String =
Path.GetFullPath(My.Resources.ResourceManager.BaseName)
        Dim localPath = myresourcefullPath.Replace(myresourcePath, "").Replace("\", "/") &
ICON_FOLDER

        htmlTemplate.AppendLine(getStringFromResources(MAP_TEMPLATE))
        mMarkerDataAsText = "["

        For i As Integer = 0 To mMarkerData.GetLength(0) - 1
            If i <> 0 Then
                mMarkerDataAsText += ","
            End If
            If dataSize = 2 Then 'lat,lon
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & "]"
            ElseIf dataSize = 3 Then 'lat,lon and title
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
```

```
& mMarkerData(i, 2) & "']"
            ElseIf dataSize = 4 Then 'lat,lon,title and image
                mMarkerDataAsText += "[" & mMarkerData(i, 0) & "," + mMarkerData(i, 1) & ",'"
& mMarkerData(i, 2) & "','" & localPath & mMarkerData(i, 3) & "']" 'Ojo a las comillas simples
en las columnas 3 y 4
            End If
        Next

        mMarkerDataAsText += "]"
        htmlTemplate.Replace(TEXT_TO_REPLACE_MARKER_DATA, mMarkerDataAsText)

        Dim tmpHtmlMapFile As String = (tmpFolder & Convert.ToString("\")) + TMP_NAME
        Dim existsMapFile As Boolean = False
        Try
            existsMapFile = createTxtFile(tmpHtmlMapFile, htmlTemplate)
        Catch ex As Exception
            MessageBox.Show("Error writing temporal file", "Writing Error",
MessageBoxButtons.OK, MessageBoxIcon.[Error])
        End Try

        If existsMapFile Then
            Return tmpHtmlMapFile
        Else
            Return Nothing
        End If
    End Function

    Private Function getMarkerDataFromPositions(pos As Double(,)) As String(,)
        Dim md As String(,) = New String(pos.GetLength(0) - 1, 1) {}
        For i As Integer = 0 To pos.GetLength(0) - 1
            md(i, 0) = pos(i, 0).ToString("g", New System.Globalization.CultureInfo("en-US"))
            md(i, 1) = pos(i, 1).ToString("g", New System.Globalization.CultureInfo("en-US"))
        Next
        Return md
    End Function

    Private Function getStringFromResources(resourceName As String) As String
        Dim assem As Assembly = Me.[GetType]().Assembly

        Using stream As Stream = assem.GetManifestResourceStream(resourceName)
            Try
                Using reader As New StreamReader(stream)
                    Return reader.ReadToEnd()
                End Using
            Catch e As Exception
                Throw New Exception((Convert.ToString("Error de acceso al Recurso '") &
resourceName) + "'" & vbCr & vbLf + e.ToString())
            End Try
        End Using
    End Function

    Private Function createTxtFile(mFile As String, content As StringBuilder) As Boolean
        Dim mPath As String = Path.GetDirectoryName(mFile)
        If Not Directory.Exists(mPath) Then
            Directory.CreateDirectory(mPath)
        End If
        If File.Exists(mFile) Then
            File.Delete(mFile)
        End If
        Dim sw As StreamWriter = File.CreateText(mFile)
        sw.Write(content.ToString())
```

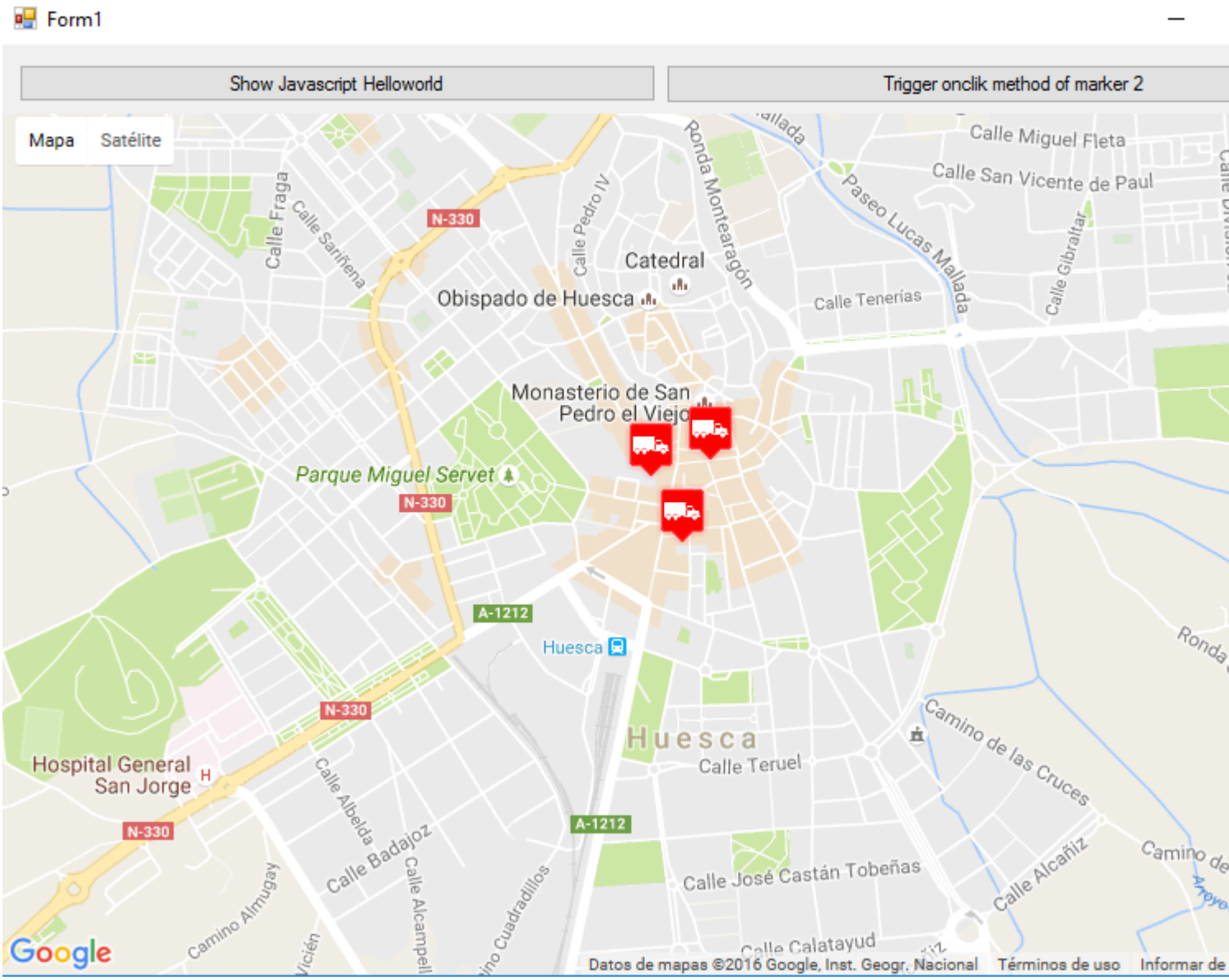
```
        sw.Close()
        Return True
    End Function
    End Class
```

Remarque: La constante MAP_TEMPLATE doit inclure le nom de votre projet

Étape 8. Maintenant, nous pouvons utiliser notre classe GoogleMapHelper pour charger la carte dans notre navigateur Web en créant et en appelant simplement sa méthode loadMap (). Comment vous construisez votre markerData est à vous. Dans cet exemple, pour clarification, je les écris à la main. Il existe 3 options pour définir les données de marqueur (voir les commentaires de la classe GoogleMapHelper). Notez que si vous utilisez la troisième option (y compris le titre et les icônes), vous devez créer un dossier nommé "marker_icons" (ou tout ce que vous définissez dans la constante ICON_FOLDER de GoogleMapHelper) dans votre dossier Debug / Release et y placer vos fichiers .png. Dans mon cas:



J'ai créé deux boutons dans mon Form1 pour illustrer la façon dont la carte et le WF interagissent. Voici à quoi ça ressemble:



Et voici le code:

Form1.vb

```
Imports System.IO
Imports System.Reflection
Imports System.Security.Permissions
Imports System.Text
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
Public Class Form1

Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    Me.wbmap.ObjectForScripting = Me

    Dim onlyPositions As Double(,) = New Double(2, 1) {{42.13557, -0.40806}, {42.13684, -
0.40884}, {42.13716, -0.40729}}
    Dim positonAndTitles As String(,) = New String(2, 2) {{"42.13557", "-0.40806", "marker0"},
{"42.13684", "-0.40884", "marker1"}, {"42.13716", "-0.40729", "marker2"}}
    Dim positonTitlesAndIcons As String(,) = New String(2, 3) {{"42.13557", "-0.40806",
"marker0", "truck_red.png"}, {"42.13684", "-0.40884", "marker1", "truck_red.png"},
```

```
{"42.13716", "-0.40729", "marker2", "truck_red.png"}}

    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, onlyPositions)
    'Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonAndTitles)
    Dim gmh As GoogleMapHelper = New GoogleMapHelper(wbmap, positonTitlesAndIcons)
    gmh.loadMap()
End Sub

'############################## CALLING JAVASCRIPT METHODS ##############################
'This methods call methods written in googlemap_template.html
Private Sub callMapJavascript(sender As Object, e As EventArgs) Handles Button1.Click
    wbmap.Document.InvokeScript("showJavascriptHelloWorld")
End Sub

Private Sub callMapJavascriptWithArguments(sender As Object, e As EventArgs) Handles
Button2.Click
    wbmap.Document.InvokeScript("focusMarkerFromIdx", New String() {2})
End Sub
'########################################################################################

'############################## METHODS CALLED FROM JAVASCRIPT ##########################
'This methods are called by the javascript defined in googlemap_template.html when some events
are triggered
Public Sub getMarkerDataFromJavascript(title As String, idx As String)
    MsgBox("Title: " & title & " idx: " & idx)
End Sub

Public Sub showVbHelloWorld()
    MsgBox("Hello world in WF from HTML")
End Sub
End Class
```

IMPORTANT: n'oubliez pas d'ajouter ces lignes avant votre définition de classe Form1:

```
<PermissionSet(SecurityAction.Demand, Name:="FullTrust")>
<System.Runtime.InteropServices.ComVisible(True)>
```

Ce qu'ils font, c'est dire au .NET Framework que nous voulons une confiance totale et rendre la classe visible pour COM afin que JavaScript soit visible sur Form1.

N'oubliez pas non plus cela dans votre fonction de chargement Form1:

```
Me.wbmap.ObjectForScripting = Me
```

Il expose votre classe Form1 au code JavaScript sur la page googlemap_template.hmtl.

Maintenant, vous pouvez exécuter et cela devrait fonctionner

################################## **COMMENT ÇA MARCHE**##############
##################

Fondamentalement, ce que fait notre classe GoogleMapHelper est de lire notre googlemap_template.html, de faire une copie temporelle, de remplacer le code associé aux marqueurs ([[MARKER_DATA]]) et d'exécuter la page dans le navigateur Web de notre formulaire. Ce fichier HTML parcourt tous les marqueurs et attribue un «clic» à chacun. Cette

fonction de clic est évidemment entièrement personnalisable. Dans l'exemple, il ouvre une infowindow si le marqueur a un titre, centre la carte dans un tel marqueur et appelle deux fonctions externes définies dans notre classe Form1.

D'autre part, nous pouvons définir d'autres fonctions javascript (avec ou sans arguments) dans ce fichier HTML à appeler depuis notre Windows Form (en utilisant wbmap.Document.InvokeScript).

Lire Google Maps dans un formulaire Windows en ligne: https://riptutorial.com/fr/vb-net/topic/5903/google-maps-dans-un-formulaire-windows

Chapitre 21: Introduction à la syntaxe

Examples

commentaires

La première chose intéressante à savoir, c'est comment écrire des commentaires.

Dans VB .NET, vous écrivez un commentaire en écrivant une apostrophe ou en écrivant `REM` . Cela signifie que le reste de la ligne ne sera pas pris en compte par le compilateur.

```
'This entire line is a comment
Dim x As Integer = 0 'This comment is here to say we give 0 value to x

REM There are no such things as multiline comments
'So we have to start everyline with the apostrophe or REM
```

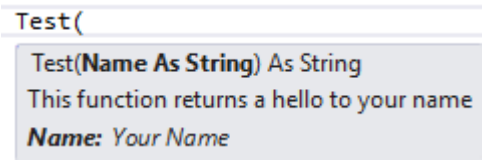
Intellisense Helper

Une chose intéressante est la possibilité d'ajouter vos propres commentaires dans Visual Studio Intellisense. Ainsi, vous pouvez rendre vos propres fonctions et classes écrites explicites. Pour ce faire, vous devez taper le symbole de commentaire trois fois la ligne au-dessus de votre fonction.

Une fois cela fait, Visual Studio ajoutera automatiquement une documentation XML:

```
''' <summary>
''' This function returns a hello to your name
''' </summary>
''' <param name="Name">Your Name</param>
''' <returns></returns>
''' <remarks></remarks>
Public Function Test(Name As String) As String
    Return "Hello " & Name
End Function
```

Après cela, si vous tapez votre fonction de test quelque part dans votre code, cette petite aide apparaîtra:



Déclaration d'une variable

Dans VB.NET, chaque variable doit être déclarée avant son utilisation (si l' option Option Explicit est définie sur **On**). Il y a deux manières de déclarer des variables:

- Dans une `Function` ou un `Sub` :

```
Dim w 'Declares a variable named w of type Object (invalid if Option Strict is On)
Dim x As String 'Declares a variable named x of type String
Dim y As Long = 45 'Declares a variable named y of type Long and assigns it the value 45
Dim z = 45 'Declares a variable named z whose type is inferred
           'from the type of the assigned value (Integer here) (if Option Infer is On)
           'otherwise the type is Object (invalid if Option Strict is On)
           'and assigns that value (45) to it
```

Voir cette réponse pour plus de détails sur `Option Explicit` , `Strict` et `Infer` .

- À l'intérieur d'une `Class` ou d'un `Module` :

Ces variables (également appelées champs dans ce contexte) seront accessibles pour chaque instance de la `Class` , ils sont déclarés. Ils pourraient être accessibles à partir de l' extérieur de la déclaration de `Class` en fonction du modificateur (`Public` , `Private` , `Protected` , `Protected Friend` ou `Friend`)

```
Private x 'Declares a private field named x of type Object (invalid if Option Strict is On)
Public y As String 'Declares a public field named y of type String
Friend z As Integer = 45 'Declares a friend field named z of type Integer and assigns it the
value 45
```

Ces champs peuvent également être déclarés avec `Dim` mais le sens change en fonction du type englobant:

```
Class SomeClass
    Dim z As Integer = 45 ' Same meaning as Private z As Integer = 45
End Class

Structure SomeStructure
    Dim y As String ' Same meaning as Public y As String
End Structure
```

Modificateurs

Les modificateurs permettent d'indiquer comment les objets externes peuvent accéder aux données d'un objet.

- Publique

Signifie que tout objet peut y accéder sans restriction

- Privé

Signifie que seul l'objet déclarant peut accéder et voir ceci

- Protégé

Signifie uniquement l'objet déclarant et tout objet qui en hérite peut y accéder et l'afficher.

- Ami

Signifie uniquement l'objet delcaring, tout objet qui en hérite et tout objet dans le même espace de noms peut y accéder et le visualiser.

```
Public Class MyClass
    Private x As Integer

    Friend Property Hello As String

    Public Sub New()
    End Sub

    Protected Function Test() As Integer
        Return 0
    End Function
End Class
```

Écrire une fonction

Une fonction est un bloc de code qui sera appelé plusieurs fois pendant l'exécution. Au lieu d'écrire le même morceau de code encore et encore, on peut écrire ce code dans une fonction et appeler cette fonction chaque fois que cela est nécessaire.

Une fonction :

- Doit être déclaré dans une *classe* ou un *module*
- Retourne une valeur (spécifiée par le type de retour)
- A un *modificateur*
- Peut prendre des paramètres pour effectuer son traitement

```
Private Function AddNumbers(X As Integer, Y As Integer) As Integer
    Return X + Y
End Function
```

Un nom de fonction, pourrait être utilisé comme déclaration de retour

```
Function sealBarTypeValidation() as Boolean
    Dim err As Boolean = False

    If rbSealBarType.SelectedValue = "" Then
        err = True
    End If

    Return err
End Function
```

est la même chose que

```
Function sealBarTypeValidation() as Boolean
    sealBarTypeValidation = False

    If rbSealBarType.SelectedValue = "" Then
        sealBarTypeValidation = True
    End If
```

```
End Function
```

Initialiseurs d'objet

- Types nommés

```
Dim someInstance As New SomeClass(argument) With {
    .Member1 = value1,
    .Member2 = value2
    '...
}
```

Est équivalent à

```
Dim someInstance As New SomeClass(argument)
someInstance.Member1 = value1
someInstance.Member2 = value2
'...
```

- Types anonymes *(l'option Infer doit être activée)*

```
 Dim anonymousInstance = New With {
     .Member1 = value1,
     .Member2 = value2
     '...
 }
```

Bien que similaire `anonymousInstance` ne pas même type que `someInstance`

Le nom du membre doit être unique dans le type anonyme et peut provenir d'une variable ou d'un autre nom de membre d'objet

```
 Dim anonymousInstance = New With {
     value1,
     value2,
     foo.value3
     '...
 }
 ' usage : anonymousInstance.value1 or anonymousInstance.value3
```

Chaque membre peut être précédé du mot clé `Key` . Ces membres seront des propriétés `ReadOnly` , ceux sans seront des propriétés de lecture / écriture

```
 Dim anonymousInstance = New With {
     Key value1,
     .Member2 = value2,
     Key .Member3 = value3
     '...
 }
```

Deux instances anonymes définies avec les mêmes membres (nom, type, présence de `Key`

et ordre) auront le même type anonyme.

```
Dim anon1 = New With { Key .Value = 10 }
Dim anon2 = New With { Key .Value = 20 }

anon1.GetType Is anon2.GetType ' True
```

Les types anonymes sont structurellement équitables. Deux instances du même type anonyme ayant au moins une propriété `Key` avec les mêmes valeurs `Key` seront égales. Vous devez utiliser la méthode `Equals` pour le tester, utiliser `=` ne compilera pas et `Is` comparera la référence de l'objet.

```
Dim anon1 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 0 }
Dim anon2 = New With { Key .Name = "Bar", Key .Age = 20, .Salary = 0 }
Dim anon3 = New With { Key .Name = "Foo", Key .Age = 10, .Salary = 10000 }

anon1.Equals(anon2) ' False
anon1.Equals(anon3) ' True although non-Key Salary isn't the same
```

Les initialiseurs de types Named et Anonymous peuvent être imbriqués et mélangés

```
Dim anonymousInstance = New With {
    value,
    Key .someInstance = New SomeClass(argument) With {
        .Member1 = value1,
        .Member2 = value2
        '...
    }
    '...
}
```

Initialiseur de collection

- Tableaux

```
Dim names = {"Foo", "Bar"} ' Inferred as String()
Dim numbers = {1, 5, 42} ' Inferred as Integer()
```

- Conteneurs (`List(Of T)` , `Dictionary(Of TKey, TValue)` , etc.)

```
Dim names As New List(Of String) From {
    "Foo",
    "Bar"
    '...
}

Dim indexedDays As New Dictionary(Of Integer, String) From {
    {0, "Sun"},
    {1, "Mon"}
    '...
}
```

Est équivalent à

```
Dim indexedDays As New Dictionary(Of Integer, String)
indexedDays.Add(0, "Sun")
indexedDays.Add(1, "Mon")
'...
```

Les éléments peuvent être le résultat d'un constructeur, d'un appel de méthode, d'un accès à une propriété. Il peut également être mélangé avec l'initialiseur d'objet.

```
Dim someList As New List(Of SomeClass) From {
    New SomeClass(argument),
    New SomeClass With { .Member = value },
    otherClass.PropertyReturningSomeClass,
    FunctionReturningSomeClass(arguments)
    '...
}
```

Il est impossible d'utiliser la syntaxe de l'initialiseur d'objet **ET la** syntaxe d'initialiseur de collection pour le même objet en même temps. Par exemple, cela **ne** fonctionnera **pas**

```
Dim numbers As New List(Of Integer) With {.Capacity = 10} _
                                   From { 1, 5, 42 }

Dim numbers As New List(Of Integer) From {
    .Capacity = 10,
    1, 5, 42
}

Dim numbers As New List(Of Integer) With {
    .Capacity = 10,
    1, 5, 42
}
```

- Type personnalisé

  Nous pouvons également autoriser la syntaxe de l'initialiseur de collection en fournissant un type personnalisé.
  Il doit implémenter `IEnumerable` et avoir une méthode accessible et compatible avec les règles de surcharge `Add` (instance, méthode d'extension partagée ou même d'extension)

  Exemple contruit:

```
Class Person
    Implements IEnumerable(Of Person) ' Inherits from IEnumerable

    Private ReadOnly relationships As List(Of Person)

    Public Sub New(name As String)
        relationships = New List(Of Person)
    End Sub

    Public Sub Add(relationName As String)
        relationships.Add(New Person(relationName))
    End Sub
```

```
    Public Iterator Function GetEnumerator() As IEnumerator(Of Person) _
        Implements IEnumerable(Of Person).GetEnumerator

        For Each relation In relationships
            Yield relation
        Next
    End Function

    Private Function IEnumerable_GetEnumerator() As IEnumerator _
        Implements IEnumerable.GetEnumerator

        Return GetEnumerator()
    End Function
End Class

' Usage
Dim somePerson As New Person("name") From {
    "FriendName",
    "CoWorkerName"
    '...
}
```

Si nous voulions ajouter l'objet Person à une `List(Of Person)` en mettant simplement le nom dans l'initialiseur de collection (mais nous ne pouvons pas modifier la classe List (Of Person)), nous pouvons utiliser une méthode d'extension

```
' Inside a Module
<Runtime.CompilerServices.Extension>
Sub Add(target As List(Of Person), name As String)
    target.Add(New Person(name))
End Sub

' Usage
Dim people As New List(Of Person) From {
    "Name1", ' no need to create Person object here
    "Name2"
}
```

Lire Introduction à la syntaxe en ligne: https://riptutorial.com/fr/vb-net/topic/3997/introduction-a-la-syntaxe

Chapitre 22: La gestion des erreurs

Examples

Essayer ... attraper ... enfin déclaration

Structure:

```
Try
    'Your program will try to run the code in this block.
    'If any exceptions are thrown, the code in the Catch Block will be executed,
    'without executing the lines after the one which caused the exception.
Catch ex As System.IO.IOException
    'If an exception occurs when processing the Try block, each Catch statement
    'is examined in textual order to determine which handles the exception.
    'For example, this Catch block handles an IOException.
Catch ex As Exception
    'This catch block handles all Exception types.
    'Details of the exception, in this case, are in the "ex" variable.
    'You can show the error in a MessageBox with the below line.
    MessageBox.Show(ex.Message)
Finally
    'A finally block is always executed, regardless of if an Exception occurred.
End Try
```

Exemple de code:

```
Try
    Dim obj = Nothing
    Dim prop = obj.Name 'This line will throw a NullReferenceException

    Console.WriteLine("Test.") ' This line will NOT be executed
Catch ex As System.IO.IOException
    ' Code that reacts to IOException.
Catch ex As NullReferenceException
    ' Code that reacts to a NullReferenceException
    Console.WriteLine("NullReferenceException: " & ex.Message)
    Console.WriteLine("Stack Trace: " & ex.StackTrace)
Catch ex As Exception
    ' Code that reacts to any other exception.
Finally
    ' This will always be run, regardless of if an exception is thrown.
    Console.WriteLine("Completed")
End Try
```

Créer une exception personnalisée et lancer

Vous pouvez créer une exception personnalisée et les lancer lors de l'exécution de votre fonction. En règle générale, vous ne devez lancer une exception que si votre fonction ne peut pas atteindre ses fonctionnalités définies.

```
Private Function OpenDatabase(Byval Server as String, Byval User as String, Byval Pwd as
```

```
String)
    if Server.trim="" then
        Throw new Exception("Server Name cannot be blank")
    elseif User.trim ="" then
        Throw new Exception("User name cannot be blank")
    elseif Pwd.trim="" then
        Throw new Exception("Password cannot be blank")
    endif

    'Here add codes for connecting to the server
End function
```

Essayez Catch dans la base de données

Vous pouvez utiliser Try..Catch pour annuler l'opération de base de données en plaçant l'instruction de restauration au niveau du segment de capture.

```
    Try
        'Do the database operation...
        xCmd.CommandText = "INSERT into ...."
        xCmd.ExecuteNonQuery()

        objTrans.Commit()
        conn.Close()
    Catch ex As Exception
        'Rollback action when something goes off
        objTrans.Rollback()
        conn.Close()
    End Try
```

L'exception non capturable

Bien que `Catch ex As Exception` prétende pouvoir gérer toutes les exceptions, il existe une exception (sans jeu de mots).

```
Imports System
Static Sub StackOverflow() ' Again no pun intended
    StackOverflow()
End Sub
Static Sub Main()
    Try
        StackOverflow()
    Catch ex As Exception
        Console.WriteLine("Exception caught!")
    Finally
        Console.WriteLine("Finally block")
    End Try
End Sub
```

Oups ... Il y a une `System.StackOverflowException` alors que la console n'a même rien imprimé! Selon MSDN ,

> À partir de .NET Framework 2.0, vous ne pouvez pas intercepter un objet StackOverflowException avec un bloc try / catch, et le processus correspondant se

termine par défaut. Par conséquent, vous devez écrire votre code pour détecter et empêcher un débordement de pile.

Donc, `System.StackOverflowException` est non capturable. Attention à ça!

Exceptions critiques

Généralement, la plupart des exceptions ne sont pas si critiques, mais il existe des exceptions très sérieuses que vous ne pourrez peut-être pas gérer, telles que la célèbre `System.StackOverflowException` . Cependant, il y en a d'autres qui pourraient être masqués par une `Catch ex As Exception` , tels que `System.OutOfMemoryException` , `System.BadImageFormatException` et `System.InvalidProgramException` . C'est une bonne pratique de programmation de les laisser si vous ne pouvez pas les gérer correctement. Pour filtrer ces exceptions, nous avons besoin d'une méthode d'assistance:

```
Public Shared Function IsCritical(ex As Exception) As Boolean
    Return TypeOf ex Is OutOfMemoryException OrElse
           TypeOf ex Is AppDomainUnloadedException OrElse
           TypeOf ex Is AccessViolationException OrElse
           TypeOf ex Is BadImageFormatException OrElse
           TypeOf ex Is CannotUnloadAppDomainException OrElse
           TypeOf ex Is ExecutionEngineException OrElse ' Obsolete one, but better to include
           TypeOf ex Is InvalidProgramException OrElse
           TypeOf ex Is System.Threading.ThreadAbortException
End Function
```

Usage:

```
Try
    SomeMethod()
Catch ex As Exception When Not IsCritical(ex)
    Console.WriteLine("Exception caught: " & ex.Message)
End Try
```

Lire La gestion des erreurs en ligne: https://riptutorial.com/fr/vb-net/topic/4232/la-gestion-des-erreurs

Chapitre 23: La gestion des fichiers

Syntaxe

- `System.IO.File.ReadAllLines(path As String)`
- `System.IO.File.ReadAllText(path As String)`
- `System.IO.File.WriteAllText(path As String, contents As String)`
- `System.IO.File.WriteAllLines(path As String, contents() As String)`

Examples

Écrire des données dans un fichier

Pour écrire le contenu d'une chaîne dans un fichier:

```
Dim toWrite As String = "This will be written to the file."
System.IO.File.WriteAllText("filename.txt", toWrite)
```

`WriteAllText` ouvre le fichier spécifié, écrit les données, puis ferme le fichier. Si le fichier cible existe, il est écrasé. Si le fichier cible n'existe pas, il est créé.

Pour écrire le contenu d'un tableau dans un fichier:

```
Dim toWrite As String() = {"This", "Is", "A", "Test"}
System.IO.File.WriteAllLines("filename.txt", toWrite)
```

`WriteAllLines` ouvre le fichier spécifié, écrit chaque valeur du tableau sur une nouvelle ligne, puis ferme le fichier. Si le fichier cible existe, il est écrasé. Si le fichier cible n'existe pas, il est créé.

Lire tout le contenu d'un fichier

Pour lire le contenu dans un fichier dans une variable de chaîne:

```
Dim fileContents As String = System.IO.File.ReadAllText("filename.txt")
```

`ReadAllText` ouvre le fichier spécifié, lit les données à la fin, puis ferme le fichier.

Pour lire un fichier, en le séparant en un élément de tableau pour chaque ligne:

```
Dim fileLines As String() = System.IO.File.ReadAllLines("filename.txt")
```

`ReadAllLines` ouvre le fichier spécifié, lit chaque ligne du fichier dans un nouvel index dans un tableau jusqu'à la fin du fichier, puis ferme le fichier.

Écrire des lignes individuellement dans un fichier texte à l'aide de

StreamWriter

```
Using sw As New System.IO.StreamWriter("path\to\file.txt")
    sw.WriteLine("Hello world")
End Using
```

L'utilisation d'un bloc `Using` est une bonne pratique lors de l'utilisation d'un objet qui implémente `IDisposable`

Lire La gestion des fichiers en ligne: https://riptutorial.com/fr/vb-net/topic/2413/la-gestion-des-fichiers

Chapitre 24: Lecture du fichier texte compressé à la volée

Examples

Lecture du fichier texte .gz ligne après ligne

Cette classe ouvre un fichier .gz (format habituel des fichiers journaux compressés) et renvoie une ligne à chaque appel de `.NextLine()`

Il n'y a pas d'utilisation de mémoire pour la décompression temporaire, très utile pour les gros fichiers.

```
Imports System.IO

Class logread_gz

  Private ptr As FileStream
  Private UnGZPtr As Compression.GZipStream
  Private line_ptr As StreamReader
  Private spath As String

  Sub New(full_filename As String)
    spath = full_filename
  End Sub

  Sub Open()
     Me.ptr = File.OpenRead(spath)
     Me.UnGZPtr = New Compression.GZipStream(ptr, Compression.CompressionMode.Decompress)
     Me.line_ptr = New StreamReader(UnGZPtr)
  End Sub()

  Function NextLine() As String
    'will return Nothing if EOF
    Return Me.line_ptr.ReadLine()
  End Function

  Sub Close()
    Me.line_ptr.Close()
    Me.line_ptr.Dispose()
    Me.UnGZPtr.Close()
    Me.UnGZPtr.Dispose()
    Me.ptr.Close()
    Me.ptr.Dispose()
  End Sub

End Class
```

Remarque: il n'y a pas de sécurité intégrée, pour des raisons de lisibilité.

Lire Lecture du fichier texte compressé à la volée en ligne: https://riptutorial.com/fr/vb-net/topic/6960/lecture-du-fichier-texte-compresse-a-la-volee

Chapitre 25: Les fonctions

Introduction

La fonction est juste comme sous. Mais la fonction renvoie une valeur. Une fonction peut accepter un ou plusieurs paramètres.

Examples

Définir une fonction

Il est très facile de définir les fonctions.

```
Function GetAreaOfARectangle(ByVal Edge1 As Integer, ByVal Edge2 As Integer) As Integer
    Return Edge1 * Edge2
End Function
```

```
Dim Area As Integer = GetAreaOfARectangle(5, 8)
Console.Writeline(Area) 'Output: 40
```

Définir une fonction # 2

```
Function Age(ByVal YourAge As Integer) As String
    Select Case YourAge
        Case Is < 18
            Return("You are younger than 18! You are teen!")
        Case 18 to 64
            Return("You are older than 18 but younger than 65! You are adult!")
        Case Is >= 65
            Return("You are older than 65! You are old!")
    End Select
End Function
```

```
Console.WriteLine(Age(48)) 'Output: You are older than 18 but younger than 65! You are adult!
```

Lire Les fonctions en ligne: https://riptutorial.com/fr/vb-net/topic/10088/les-fonctions

Chapitre 26: Les opérateurs

Remarques

Les opérateurs sont utilisés pour attribuer ou comparer des valeurs. Ils se composent d'un seul symbole ou mot-clé et sont généralement pris en sandwich entre une valeur gauche et une valeur droite. Par exemple: `right = left` .

Les opérateurs sont intrinsèques à la langue (tels que `=`) et non à des fonctions telles que celles fournies par System.Math.

Examples

Comparaison

Les opérateurs de comparaison comparent deux valeurs et vous renvoient un booléen (`True` ou `False`) comme résultat.

Égalité

- Le signe égal `=` est utilisé à la fois pour la comparaison d'égalité et pour l'affectation.
  `If leftValue = rightValue Then` ...

Inégalité

- Le support d'angle gauche de la cornière à angle droit `<>` effectue une comparaison inégale.
  `If leftValue <> rightValue Then` ...

Plus grand que

- La cornière gauche `<` effectue une comparaison supérieure.
  `If leftValue < rightValue Then` ...

Meilleur que ou égal

- Le signe d'égalité égal au crochet d'angle gauche `=>` effectue une comparaison supérieure ou égale.
  `If leftValue =< rightValue Then` ...

Moins que

- La cornière droite `>` effectue une comparaison inférieure.
  `If leftValue > rightValue Then` ...

Moins qu'égale

- Le nid de signe égal au crochet à angle droit `=>` effectue une comparaison supérieure ou

égale.

`If leftValue => rightValue Then ...`

Comme

- L'opérateur `Like` teste l'égalité d'une chaîne et d'un modèle de recherche.
- L'opérateur `Like` s'appuie sur l' énoncé de comparaison d'options
- Le tableau suivant répertorie les modèles disponibles. Source: https://msdn.microsoft.com/en-us/library/swf8kaxw.aspx (section Remarques)

Caractères dans le *motif*	Allumettes dans la *ficelle*
?	N'importe quel caractère
*	Zéro ou plusieurs caractères
#	N'importe quel chiffre (0 - 9)
[charlist]	N'importe quel caractère dans la *charlist*
[! charlist]	N'importe quel caractère ne *figurant* pas dans *charlist*

- Voir plus d'informations sur MSDN dans la section remarques.
  `If string Like pattern Then ...`

Affectation

Il y a un seul opérateur d'affectation dans VB.

- Le signe égal `=` est utilisé à la fois pour la comparaison d'égalité et pour l'affectation.
  `Dim value = 5`

Remarques
Méfiez-vous des comparaisons entre égalité et cession.

```
Dim result = leftValue = rightValue
```

Dans cet exemple, vous pouvez voir que le signe égal est utilisé à la fois comme opérateur de comparaison et comme opérateur d'affectation, contrairement aux autres langages. Dans ce cas, `result` sera de type `Boolean` et contiendra la valeur de la comparaison d'égalité entre `leftValue` et `rightValue` .

Connexe: Utilisation de Option Strict On pour déclarer les variables correctement

Math

Si vous avez les variables suivantes

```
Dim leftValue As Integer = 5
```

```
Dim rightValue As Integer = 2
Dim value As Integer = 0
```

Ajout effectué par le signe plus `+` .

```
value  = leftValue + rightValue

'Output the following:
'7
```

Soustraction Effectuée par le signe moins `-` .

```
value = leftValue - rightValue

'Output the following:
'3
```

Multiplication Effectuée par le symbole étoile `*` .

```
value = leftValue * rightValue

'Output the following:
'10
```

Division Effectuée par le symbole barre oblique `/` .

```
value = leftValue / rightValue

'Output the following:
'2.5
```

Division entière Effectué par le symbole barre oblique inverse `\` .

```
value = leftValue \ rightValue

'Output the following:
'2
```

Module exécuté par le mot clé `Mod` .

```
value = leftValue Mod rightValue

'Output the following:
'1
```

Augmenter à une puissance de performé par le symbole `^` .

```
value = leftValue ^ rightValue

'Output the following:
'25
```

Élargissement et Rétrécissement

Besoin de retouche

Surcharge de l'opérateur

Besoin de retouche

Bit à bit

Ce sont les opérateurs binaires dans VB.NET: Et, Ou, Xor, Pas

Exemple d'opération par bit

```
Dim a as Integer
a = 3 And 5
```

La valeur de a sera 1. Le résultat est obtenu après avoir comparé 3 et 5 en binaire pour. 3 sous forme binaire est 011 et 5 sous forme binaire est 101. L'opérateur And place 1 si les deux bits sont 1. Si l'un des bits est 0 alors la valeur sera 0

```
3 And 5 will be  011
                 101
                 ---
                 001
```

Donc, le résultat binaire est 001 et quand il est converti en décimal, la réponse sera 1.

Ou l'opérateur place 1 si les deux ou un bit est 1

```
   3 Or 5 will be 011
                  101
                  ---
                  111
```

L'opérateur Xor place 1 si un seul bit est 1 (pas les deux)

```
   3 Xor 5 will be  011
                    101
                    ---
                    110
```

Pas opérateur retourne les bits, y compris le signe

```
Not 5 will be - 010
```

Concaténation de chaînes

La concaténation de chaînes consiste à combiner plusieurs chaînes en une seule chaîne.

La concaténation de chaînes est effectuée avec le symbole `&` .

```
Dim one As String = "Hello "
Dim two As String = "there"
Dim result As String = one & two
```

Les valeurs non-chaîne seront converties en chaîne lors de l'utilisation de `&` .

```
Dim result as String = "2" & 10 ' result  = "210"
```

Utilisez toujours `&` (esperluette) pour effectuer la concaténation de chaîne.

NE PAS FAIRE CE
Bien qu'il soit possible, dans les cas les *plus simples* , d'utiliser le symbole `+` pour faire de la concaténation de chaînes, vous ne devriez jamais le faire. Si un côté du symbole plus n'est pas une chaîne, lorsque Option strict est désactivé, le comportement devient non intuitif, lorsque Option strict est activé, il génère une erreur de compilation. Considérer:

```
Dim value = "2" + 10    ' result = 12  (data type Double)
Dim value = "2" + "10"  ' result = "210"  (data type String)
Dim value = "2g" + 10   ' runtime error
```

Le problème ici est que si l'opérateur `+` voit un opérande quelconque qui est un type numérique, il supposera que le programmeur a voulu effectuer une opération arithmétique et tenter de convertir l'autre opérande en un type numérique équivalent. Dans les cas où l'autre opérande est une chaîne contenant un nombre (par exemple, "10"), la chaîne est *convertie en un nombre* , puis *arithmétiquement* ajoutée à l'autre opérande. Si l'autre opérande ne peut pas être converti en un nombre (par exemple, "2g"), l'opération sera bloquée en raison d'une erreur de conversion de données. L'opérateur `+` n'effectuera la concaténation de chaîne que si les *deux* opérandes sont de type `String` .

L'opérateur `&` , cependant, est conçu pour la concaténation de chaînes et convertit les types de non-chaînes en chaînes.

Lire Les opérateurs en ligne: https://riptutorial.com/fr/vb-net/topic/3257/les-operateurs

Chapitre 27: Liaison de données WPF XAML

Introduction

Cet exemple montre comment créer un ViewModel et un View dans le modèle MVVM et WPF, et comment lier les deux ensemble, de sorte que chacun soit mis à jour chaque fois que l'autre est modifié.

Examples

Liaison d'une chaîne dans le ViewModel à une TextBox dans la vue

SampleViewModel.vb

```
'Import classes related to WPF for simplicity
Imports System.Collections.ObjectModel
Imports System.ComponentModel

Public Class SampleViewModel
    Inherits DependencyObject
    'A class acting as a ViewModel must inherit from DependencyObject

    'A simple string property
    Public Property SampleString as String
        Get
            Return CType(GetValue(SampleStringProperty), String)
        End Get

        Set(ByVal value as String)
            SetValue(SampleStringProperty, value)
        End Set
    End Property

    'The DependencyProperty that makes databinding actually work
    'for the string above
    Public Shared ReadOnly SampleStringProperty As DependencyProperty = _
                           DependencyProperty.Register("SampleString", _
                           GetType(String), GetType(SampleViewModel), _
                           New PropertyMetadata(Nothing))

End Class
```

Un DependencyProperty peut être facilement ajouté à l'aide de l' `wpfdp` code `wpfdp` (tapez `wpfdp` , puis appuyez deux fois sur la touche `TAB`), mais l'extrait de code n'est pas sûr et ne sera pas compilé sous `Option Strict On` .

SampleWindow.xaml

```
<Window x:Class="SampleWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:des="http://schemas.microsoft.com/expression/blend/2008"
        DataContext="{Binding}"
        Loaded="Window_Loaded">
    <Grid>
        <TextBox>
            <TextBox.Text>
                <Binding Path="SampleString" />
            </TextBox.Text>
        </TextBox>
    </Grid>
</Window>
```

SampleWindow.xaml.vb

```
Class SampleWindow

    Private WithEvents myViewModel As New SampleViewModel()

    Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
        Me.DataContext = myViewModel
    End Sub
End Class
```

Notez que c'est un moyen très rudimentaire d'implémenter MVVM et la liaison de données. Une pratique plus robuste consisterait à utiliser une plate-forme comme Unity pour «injecter» le ViewModel dans la vue.

Lire Liaison de données WPF XAML en ligne: https://riptutorial.com/fr/vb-net/topic/8177/liaison-de-donnees-wpf-xaml

Chapitre 28: LINQ

Introduction

LINQ (Language Integrated Query) est une expression qui extrait des données d'une source de données. LINQ simplifie cette situation en proposant un modèle cohérent pour travailler avec des données sur différents types de sources de données et de formats. Dans une requête LINQ, vous travaillez toujours avec des objets. Vous utilisez les mêmes modèles de codage de base pour interroger et transformer des données dans des documents XML, des bases de données SQL, des ensembles de données ADO.NET, des collections .NET et tout autre format pour lequel un fournisseur LINQ est disponible.

Examples

Projection

```
' sample data
Dim sample = {1, 2, 3, 4, 5}

' using "query syntax"
Dim squares = From number In sample Select number * number

' same thing using "method syntax"
Dim squares = sample.Select (Function (number) number * number)
```

Nous pouvons projeter plusieurs résultats à la fois aussi

```
Dim numbersAndSquares =
    From number In sample Select number, square = number * number

Dim numbersAndSquares =
    sample.Select (Function (number) New With {Key number, Key .square = number * number})
```

Sélection à partir d'un tableau avec une condition simple

```
Dim sites() As String = {"Stack Overflow", "Super User", "Ask Ubuntu", "Hardware
Recommendations"}
Dim query = From x In sites Where x.StartsWith("S")
' result = "Stack Overflow", "Super User"
```

Query sera un objet énumérable contenant `Stack Overflow` et `Super User` . `x` dans la requête est une variable itérative où sera stocké chaque objet vérifié par la clause `Where` .

Tableau de mappage par clause Select

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
```

```
                          "Ask Ubuntu",
                          "Hardware Recommendations"}
Dim query = From x In sites Select x.Length
' result = 14, 10, 10, 24
```

Le résultat de la requête sera un objet énumérable contenant des longueurs de chaînes dans le tableau d'entrée. Dans cet exemple, ce serait les valeurs 14, 10, 10, 24. x dans la requête est une variable itérative où sera stocké chaque objet du tableau d'entrée.

Sortie de commande

```
Dim sites() As String = {"Stack Overflow",
                         "Super User",
                         "Ask Ubuntu",
                         "Hardware Recommendations"}

Dim query = From x In sites
            Order By x.Length

' result = "Super User", "Ask Ubuntu", "Stack Overflow", "Hardware Recommendations"
```

La clause OrderBy classe la sortie par la valeur renvoyée par la clause. Dans cet exemple, il s'agit de Longueur de chaque chaîne. L'ordre de sortie par défaut est croissant. Si vous avez besoin de descendre, vous pouvez spécifier une clause après un mot clé `Descending` .

```
Dim query = From x In sites
            Order By x.Length Descending
```

Générer un dictionnaire à partir de IEnumerable

```
' Just setting up the example
Public Class A
    Public Property ID as integer
    Public Property Name as string
    Public Property OtherValue as Object
End Class

Public Sub Example()
    'Setup the list of items
    Dim originalList As New List(Of A)
    originalList.Add(New A() With {.ID = 1, .Name = "Item 1", .OtherValue = "Item 1 Value"})
    originalList.Add(New A() With {.ID = 2, .Name = "Item 2", .OtherValue = "Item 2 Value"})
    originalList.Add(New A() With {.ID = 3, .Name = "Item 3", .OtherValue = "Item 3 Value"})

    'Convert the list to a dictionary based on the ID
    Dim dict As Dictionary(Of Integer, A) = originalList.ToDictionary(function(c) c.ID,
function(c) c)

    'Access Values From The Dictionary
    console.Write(dict(1).Name) ' Prints "Item 1"
    console.Write(dict(1).OtherValue) ' Prints "Item 1 Value"
End Sub
```

Obtenir des valeurs distinctes (en utilisant la méthode Distinct)

```
Dim duplicateFruits = New List(Of String) From {"Grape", "Apple", "Grape", "Apple", "Grape"}
'At this point, duplicateFruits.Length = 5

Dim uniqueFruits = duplicateFruits.Distinct();
'Now, uniqueFruits.Count() = 2
'If iterated over at this point, it will contain 1 each of "Grape" and "Apple"
```

Lire LINQ en ligne: https://riptutorial.com/fr/vb-net/topic/3111/linq

Chapitre 29: Manipulation de connexion

Examples

Propriété de connexion publique

```
Imports System.Data.OleDb

Private WithEvents _connection As OleDbConnection
Private _connectionString As String = "myConnectionString"

Public ReadOnly Property Connection As OleDbConnection
    Get
        If _connection Is Nothing Then
            _connection = New OleDbConnection(_connectionString)
            _connection.Open()
        Else
            If _connection.State <> ConnectionState.Open Then
                _connection.Open()
            End If
        End If
        Return _connection
    End Get
End Property
```

Lire Manipulation de connexion en ligne: https://riptutorial.com/fr/vb-net/topic/6398/manipulation-de-connexion

Chapitre 30: Méthodes d'extension

Remarques

Les méthodes d'extension sont des méthodes (`Sub` ou `Function`) qui ajoutent des fonctionnalités à un type (qui peut être un type de référence ou un type de valeur). Ces Types peuvent ou non vous appartenir.

Ils peuvent ou peuvent ne pas être dans le même assemblage que le type qu'ils sont destinés à modifier. Vous pouvez autoriser un opt-in à vos méthodes d'extension en les isolant dans leur propre espace de noms. Ou, si vous préférez, vous pouvez les rendre toujours disponibles en les incluant dans le même espace-nom que le type qu'ils modifient (en supposant que toutes les références d'assemblage sont en place et correctes). Voir le projet Entity Framework Core 1.0 sur GitHub pour un bon exemple du style opt-in des méthodes d'extension.

Les méthodes d'extension dans VB ont quelques exigences:

- Les méthodes d'extension ne peuvent être déclarées que dans des modules.
- Les méthodes d'extension doivent être décorées avec l'attribut `Extension()` .
- L'espace de noms ExtensionAttribute doit être disponible dans votre module.
  `Imports System.Runtime.CompilerServices`
- Le premier paramètre de la méthode doit être d'un type auquel cette méthode sera associée.
- Le premier paramètre de la méthode représentera l'instance sur laquelle cette méthode fonctionne. (Équivalent à `Me` si c'était une méthode d'instance réelle).
- Une méthode d'extension peut être appelée comme méthode régulière en fournissant tous les paramètres si elle n'est pas appelée sur l'objet instancié.

Examples

Créer une méthode d'extension

Les méthodes d'extension sont utiles pour étendre le comportement des bibliothèques que nous ne possédons pas.

Ils sont utilisés de manière similaire aux méthodes d'instance grâce au sucre syntaxique du compilateur:

```
Sub Main()
    Dim stringBuilder = new StringBuilder()

    'Extension called directly on the object.
    stringBuilder.AppendIf(true, "Condition was true")

    'Extension called as a regular method. This defeats the purpose
    'of an extension method but should be noted that it is possible.
    AppendIf(stringBuilder, true, "Condition was true")

End Sub
```

```
<Extension>
Public Function AppendIf(stringBuilder As StringBuilder, condition As Boolean, text As String)
As StringBuilder
    If(condition) Then stringBuilder.Append(text)

    Return stringBuilder
End Function
```

Pour avoir une méthode d'extension utilisable, la méthode nécessite l'attribut `Extension` et doit être déclarée dans un `Module` .

Rendre le langage plus fonctionnel avec les méthodes d'extension

Un bon usage de la méthode d'extension est de rendre le langage plus fonctionnel

```
Sub Main()
    Dim strings = { "One", "Two", "Three" }

    strings.Join(Environment.NewLine).Print()
End Sub

<Extension>
Public Function Join(strings As IEnumerable(Of String), separator As String) As String
    Return String.Join(separator, strings)
End Function

<Extension>
Public Sub Print(text As String)
    Console.WriteLine(text)
End Sub
```

Rembourrage numérique

```
Public Module Usage
  Public Sub LikeThis()
    Dim iCount As Integer
    Dim sCount As String

    iCount = 245
    sCount = iCount.PadLeft(4, "0")

    Console.WriteLine(sCount)
    Console.ReadKey()
  End Sub
End Module



Public Module Padding
  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer) As String
    Return Value.PadLeft(Length, Space(Length))
  End Function
```

```
  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer) As String
    Return Value.PadRight(Length, Space(Length))
  End Function



  <Extension>
  Public Function PadLeft(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadLeft(Length, Character)
  End Function



  <Extension>
  Public Function PadRight(Value As Integer, Length As Integer, Character As Char) As String
    Return CStr(Value).PadRight(Length, Character)
  End Function
End Module
```

Obtenir la version d'assemblage à partir d'un nom fort

Exemple d'appeler une méthode d'extension en tant qu'extension et méthode régulière.

```
public Class MyClass
  Sub Main()

        'Extension called directly on the object.
        Dim Version = Assembly.GetExecutingAssembly.GetVersionFromAssembly()

        'Called as a regular method.
        Dim Ver = GetVersionFromAssembly(SomeOtherAssembly)

    End Sub
End Class
```

La méthode d'extension dans un module. Rendez le module Public si les extensions sont compilées dans une DLL et seront référencées dans un autre assembly.

```
Public Module Extensions
    ''' <summary>
    ''' Returns the version number from the specified assembly using the assembly's strong
name.
    ''' </summary>
    ''' <param name="Assy">[Assembly] Assembly to get the version info from.</param>
    ''' <returns>[String]</returns>
    <Extension>
    Friend Function GetVersionFromAssembly(ByVal Assy As Assembly) As String
        Return Split(Split(Assy.FullName,",")(1),"=")(1)
    End Function
End Module
```

Lire Méthodes d'extension en ligne: https://riptutorial.com/fr/vb-net/topic/1592/methodes-d-extension

Chapitre 31: Modèle asynchrone basé sur des tâches

Examples

Utilisation basique d'Async / Await

Vous pouvez démarrer un processus lent en parallèle, puis collecter les résultats une fois terminés:

```
Public Sub Main()
    Dim results = Task.WhenAll(SlowCalculation, AnotherSlowCalculation).Result

    For Each result In results
        Console.WriteLine(result)
    Next
End Sub

Async Function SlowCalculation() As Task(Of Integer)
     Await Task.Delay(2000)

     Return 40
End Function

Async Function AnotherSlowCalculation() As Task(Of Integer)
    Await Task.Delay(2000)

    Return 60
End Function
```

Après deux secondes, les résultats seront disponibles.

Utiliser TAP avec LINQ

Vous pouvez créer un `IEnumerable` de `Task` en transmettant `AddressOf AsyncMethod` à la méthode **LINQ** `Select` , puis démarrez et attendez tous les résultats avec `Task.WhenAll`

Si votre méthode a des paramètres correspondant à l'appel **LINQ** précédent, ils seront automatiquement mappés.

```
Public Sub Main()
    Dim tasks = Enumerable.Range(0, 100).Select(AddressOf TurnSlowlyIntegerIntoString)

    Dim resultingStrings = Task.WhenAll(tasks).Result

    For Each value In resultingStrings
        Console.WriteLine(value)
    Next
End Sub

Async Function TurnSlowlyIntegerIntoString(input As Integer) As Task(Of String)
```

```
    Await Task.Delay(2000)

    Return input.ToString()
End Function
```

Pour mapper différents arguments, vous pouvez remplacer la `AddressOf Method` par un lambda:

```
Function(linqData As Integer) MyNonMatchingMethod(linqData, "Other parameter")
```

Lire Modèle asynchrone basé sur des tâches en ligne: https://riptutorial.com/fr/vb-net/topic/2936/modele-asynchrone-base-sur-des-taches

Chapitre 32: Mots-clés ByVal et ByRef

Examples

Mot clé ByVal

Le mot-clé ByVal avant le paramètre method (ou aucun mot-clé tel que ByVal est supposé par défaut) indique que ce paramètre sera envoyé de manière à **ne pas** permettre à la méthode de modifier (affecter une nouvelle valeur) la variable sous-jacente au paramètre.
Cela n'empêche pas le contenu (ou l'état) de l'argument d'être modifié s'il s'agit d'une classe.

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}

        Foo (someInstance)
        ' here someInstance is not Nothing (still the same object)
        ' but someInstance.Member is -42 (internal state can still be changed)

        Dim number As Integer = 42
        Foo(number)
        ' here number is still 42
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Foo(arg As Integer) ' No ByVal or ByRef keyword, ByVal is assumed
        arg = -arg
    End Sub
End Module
```

Mot-clé ByRef

Le mot-clé ByRef avant le paramètre de méthode indique que ce paramètre sera envoyé de manière à permettre à la méthode de modifier (attribuer une nouvelle valeur) la variable sous-jacente au paramètre.

```
Class SomeClass
    Public Property Member As Integer
End Class

Module Program
    Sub Main()
        Dim someInstance As New SomeClass With {.Member = 42}
```

```
        Foo (someInstance)
        ' here someInstance is not Nothing
        ' but someInstance.Member is -42

        Bar(someInstance)
        ' here someInstance is Nothing
    End Sub

    Sub Foo(ByVal arg As SomeClass)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub

    Sub Bar(ByRef param As Integer)
        arg.Member = -arg.Member ' change argument content
        arg = Nothing ' change (re-assign) argument
    End Sub
End Module
```

Lire Mots-clés ByVal et ByRef en ligne: https://riptutorial.com/fr/vb-net/topic/4653/mots-cles-byval-et-byref

Chapitre 33: Mots-clés OOP

Examples

Définir une classe

Les cours sont des aspects essentiels de la POO. Une classe est comme le "plan" d'un objet. Un objet possède les propriétés d'une classe, mais les caractéristiques ne sont pas définies dans la classe elle-même. Chaque objet pouvant être différent, ils définissent leurs propres caractéristiques.

```
Public Class Person
End Class

Public Class Customer
End Class
```

Une classe peut également contenir des *sous-classes* . Une sous-classe hérite des mêmes propriétés et comportements que sa classe parente, mais peut avoir ses propres propriétés et classes.

Modificateurs d'héritage (sur les classes)

Héritiers

Spécifie la classe de base (ou parent)

```
Public Class Person
End Class

Public Class Customer
    Inherits Person

End Class

'One line notation
Public Class Student : Inherits Person
End Class
```

Objets possibles:

```
Dim p As New Person
Dim c As New Customer
Dim s As New Student
```

Non Héritable

Empêche les programmeurs d'utiliser la classe en tant que classe de base.

```
Public NotInheritable Class Person
End Class
```

Objets possibles:

```
Dim p As New Person
```

MustInherit

Spécifie que la classe est destinée à être utilisée comme classe de base uniquement. (Classe abstraite)

```
Public MustInherit Class Person
End Class

Public Class Customer
    Inherits Person
End Class
```

Objets possibles:

```
Dim c As New Customer
```

Modificateurs d'héritage (sur les propriétés et les méthodes)

Irrégulable

Permet à une propriété ou à une méthode d'une classe d'être remplacée dans une classe dérivée.

```
Public Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class
```

Remplace

Substitue une propriété ou une méthode Overridable définie dans la classe de base.

```
Public Class Customer
    Inherits Person

    'Base Class must be Overridable
    Public Overrides Sub DoSomething()
```

```
        Console.WriteLine("Customer")
    End Sub
End Class
```

NotOverridable

Empêche une propriété ou une méthode d'être remplacée dans une classe héritée.
Comportement par défaut Ne peut être déclaré que sur les **méthodes de remplacement**

```
Public Class Person

    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

End Class

Public Class Customer
    Inherits Person

    Public NotOverridable Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class

Public Class DetailedCustomer
    Inherits Customer

    'DoSomething can't be overridden
End Class
```

Exemple d'utilisation:

```
Dim p As New Person
p.DoSomething()

Dim c As New Customer
c.DoSomething()

Dim d As New DetailedCustomer
d.DoSomething()
```

Sortie:

```
Person
Customer
Customer
```

MustOverride

Exige qu'une classe dérivée remplace la propriété ou la méthode.

Les méthodes MustOverride doivent être déclarées dans les **classes MustInherit.**

```
Public MustInherit Class Person

    Public MustOverride Sub DoSomething()
    'No method definition here

End Class

Public Class Customer
    Inherits Person

    'DoSomething must be overridden
    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Exemple d'utilisation:

```
Dim c As New Customer
c.DoSomething()
```

Sortie:

```
Customer
```

MyBase

Le mot clé MyBase se comporte comme une variable objet faisant référence à la classe de base de l'instance actuelle d'une classe.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub
End Class

Public Class Customer
    Inherits Person

    Public Sub DoSomethingElse()
        MyBase.DoSomething()
    End Sub

End Class
```

Exemple d'utilisation:

```
Dim p As New Person
p.DoSomething()
```

```
Console.WriteLine("----")

Dim c As New Customer
c.DoSomething()
c.DoSomethingElse()
```

Sortie:

```
Person
----
Person
Person
```

Moi vs MyClass

Me utilise l'instance d'objet en cours.

MyClass utilise la définition de membre dans la classe où le membre est appelé

```
Class Person
    Public Overridable Sub DoSomething()
        Console.WriteLine("Person")
    End Sub

    Public Sub useMe()
        Me.DoSomething()
    End Sub

    Public Sub useMyClass()
        MyClass.DoSomething()
    End Sub
End Class

Class Customer
    Inherits Person

    Public Overrides Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub
End Class
```

Exemple d'utilisation:

```
Dim c As New Customer
c.useMe()
c.useMyClass()
```

Sortie:

```
Customer
Person
```

Surcharge

La surcharge est la création de plusieurs procédures, constructeurs d'instance ou propriétés dans une classe avec le même nom mais différents types d'arguments.

```
Class Person
    Overloads Sub Display(ByVal theChar As Char)
        ' Add code that displays Char data.
    End Sub

    Overloads Sub Display(ByVal theInteger As Integer)
        ' Add code that displays Integer data.
    End Sub

    Overloads Sub Display(ByVal theDouble As Double)
        ' Add code that displays Double data.
    End Sub
End Class
```

Ombres

Il redéfinit un membre qui n'est pas remplaçable. Seuls les appels à l'instance seront affectés. Le code à l'intérieur des classes de base ne sera pas affecté par ceci.

```
Public Class Person
    Public Sub DoSomething()
        Console.WriteLine("Person")
    End Sub


    Public Sub UseMe()
        Me.DoSomething()
    End Sub
End Class
Public Class Customer
    Inherits Person
    Public Shadows Sub DoSomething()
        Console.WriteLine("Customer")
    End Sub

End Class
```

Exemple d'utilisation:

```
Dim p As New Person
Dim c As New Customer
p.UseMe()
c.UseMe()
Console.WriteLine("----")
p.DoSomething()
c.DoSomething()
```

Sortie:

```
Person
Person
----
```

```
Person
Customer
```

Pièges :

Example1, Créer un nouvel objet via un générique. Quelle fonction sera utilisée ??

```
Public Sub CreateAndDoSomething(Of T As {Person, New})()
    Dim obj As New T
    obj.DoSomething()
End Sub
```

exemple d'utilisation:

```
Dim p As New Person
p.DoSomething()
Dim s As New Student
s.DoSomething()
Console.WriteLine("----")
CreateAndDoSomething(Of Person)()
CreateAndDoSomething(Of Student)()
```

Sortie: Par intuition, le résultat devrait être le même. Pourtant, ce n'est pas vrai.

```
Person
Student
----
Person
Person
```

Exemple 2:

```
Dim p As Person
Dim s As New Student
p = s
p.DoSomething()
s.DoSomething()
```

Résultat: Par intuition, vous pourriez penser que p et s sont égaux et se comporteront de la même manière. Pourtant, ce n'est pas vrai.

```
Person
Student
```

Dans ces exemples simples, il est facile d'apprendre le comportement étrange de Shadows. Mais dans la vraie vie, cela apporte beaucoup de surprises. Il est conseillé d'éviter l'utilisation des ombres. On devrait utiliser d'autres alternatives autant que possible (dérogations, etc.)

Interfaces

```
Public Interface IPerson
    Sub DoSomething()
End Interface

Public Class Customer
    Implements IPerson
    Public Sub DoSomething() Implements IPerson.DoSomething
        Console.WriteLine("Customer")
    End Sub

End Class
```

Lire Mots-clés OOP en ligne: https://riptutorial.com/fr/vb-net/topic/4273/mots-cles-oop

Chapitre 34: Multithreading

Examples

Multithreading en utilisant la classe de thread

Cet exemple utilise la classe de `Thread` , mais des applications multithread peuvent également être effectuées à l'aide de `BackgroundWorker` . Les fonctions `AddNumber` , `SubstractNumber` et `DivideNumber` seront exécutées par des threads distincts:

Modifier: le thread d'interface utilisateur attend que les fils enfants se terminent et affiche le résultat.

```
Module Module1
    'Declare the Thread and assign a sub to that
    Dim AddThread As New Threading.Thread(AddressOf AddNumber)
    Dim SubstractThread As New Threading.Thread(AddressOf SubstractNumber)
    Dim DivideThread As New Threading.Thread(AddressOf DivideNumber)

    'Declare the variable for holding the result
    Dim addResult As Integer
    Dim SubStractResult As Integer
    Dim DivisionResult As Double

    Dim bFinishAddition As Boolean = False
    Dim bFinishSubstration As Boolean = False
    Dim bFinishDivision As Boolean = False

    Dim bShownAdditionResult As Boolean = False
    Dim bShownDivisionResult As Boolean = False
    Dim bShownSubstractionResult As Boolean = False

    Sub Main()

        'Now start the trheads
        AddThread.Start()
        SubstractThread.Start()
        DivideThread.Start()

        'Wait and display the results in console
        Console.WriteLine("Waiting for threads to finish...")
        Console.WriteLine("")

        While bFinishAddition = False Or bFinishDivision = False Or bFinishSubstration = False
            Threading.Thread.Sleep(50)     'UI thread is sleeping
            If bFinishAddition And Not bShownAdditionResult Then
                Console.WriteLine("Addition Result : " & addResult)
                bShownAdditionResult = True
            End If

            If bFinishSubstration And Not bShownSubstractionResult Then
                Console.WriteLine("Substraction Result : " & SubStractResult)
                bShownSubstractionResult = True
            End If
```

```
            If bFinishDivision  And Not bShownDivisionResult Then
                Console.WriteLine("Division Result : " & DivisionResult)
                bShownDivisionResult = True
            End If

        End While

        Console.WriteLine("")
        Console.WriteLine("Finished all threads.")
        Console.ReadKey()
    End Sub


    Private Sub AddNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 100
            addResult = addResult + (n1 + n2)
            Threading.Thread.Sleep(50)      'sleeping Add thread
        Next
        bFinishAddition = True
    End Sub

    Private Sub SubstractNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11

        For i As Integer = 0 To 80
            SubStractResult = SubStractResult - (n1 - n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishSubstration = True
    End Sub

    Private Sub DivideNumber()
        Dim n1 As Integer = 22
        Dim n2 As Integer = 11
        For i As Integer = 0 To 60
            DivisionResult = DivisionResult + (n1 / n2)
            Threading.Thread.Sleep(50)
        Next
        bFinishDivision = True
    End Sub

End Module
```

Lire Multithreading en ligne: https://riptutorial.com/fr/vb-net/topic/6756/multithreading

Chapitre 35: NullReferenceException

Remarques

NullReferenceException est renvoyé chaque fois qu'une variable est vide et que l'une de ses méthodes / propriétés est référencée. Pour éviter cela, assurez-vous que toutes les variables sont initialisées correctement (`new` opérateur) et que toutes les méthodes renvoient une valeur non nulle.

Examples

Variable non initialisée

Mauvais code

```
Dim f As System.Windows.Forms.Form
f.ShowModal()
```

BON CODE

```
Dim f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Dim f As New System.Windows.Forms.Form ' alternative syntax
f.ShowModal()
```

MÊME CODE MEILLEUR (Assurer une élimination correcte de l'objet IDisposable plus d'informations)

```
Using f As System.Windows.Forms.Form = New System.Windows.Forms.Form
' Using f As New System.Windows.Forms.Form ' alternative syntax
    f.ShowModal()
End Using
```

Retour vide

```
Function TestFunction() As TestClass
    Return Nothing
End Function
```

Mauvais code

```
TestFunction().TestMethod()
```

BON CODE

```
Dim x = TestFunction()
If x IsNot Nothing Then x.TestMethod()
```

14.0

Opérateur conditionnel nul

```
TestFunction()?.TestMethod()
```

Lire NullReferenceException en ligne: https://riptutorial.com/fr/vb-net/topic/4076/nullreferenceexception

Chapitre 36: Objets jetables

Examples

Concept de base d'IDisposable

Chaque fois que vous instanciez une classe qui implémente `IDisposable` , vous devez appeler `.Dispose` [1] sur cette classe lorsque vous avez fini de l'utiliser. Cela permet à la classe de nettoyer toutes les dépendances gérées ou non gérées qu'elle peut utiliser. Ne pas le faire pourrait provoquer une fuite de mémoire.

Le mot-clé `Using` s'assure que `.Dispose` est appelé, sans que vous ayez à *l'* appeler *explicitement* .

Par exemple sans `Using` :

```
Dim sr As New StreamReader("C:\foo.txt")
Dim line = sr.ReadLine
sr.Dispose()
```

Maintenant avec l' `Using` :

```
Using sr As New StreamReader("C:\foo.txt")
    Dim line = sr.ReadLine
End Using '.Dispose is called here for you
```

L'un des principaux avantages de l' `Using` has est la levée d'une exception, car elle *garantit que* `.Dispose` est appelée.

Considérer ce qui suit. Si une exception est levée, vous devez vous rappeler d'appeler .Dispose mais vous devrez peut-être aussi vérifier l'état de l'objet pour vous assurer que vous n'obtenez pas d'erreur de référence nulle, etc.

```
    Dim sr As StreamReader = Nothing
    Try
        sr = New StreamReader("C:\foo.txt")
        Dim line = sr.ReadLine
    Catch ex As Exception
        'Handle the Exception
    Finally
        If sr IsNot Nothing Then sr.Dispose()
    End Try
```

Un bloc d'utilisation signifie que vous n'avez pas à vous en souvenir et que vous pouvez déclarer votre objet dans l' `try` :

```
    Try
        Using sr As New StreamReader("C:\foo.txt")
            Dim line = sr.ReadLine
        End Using
```

```
    Catch ex As Exception
        'sr is disposed at this point
    End Try
```

---

[1] Dois-je toujours appeler Dispose () sur mes objets DbContext? Nan

Déclarer plus d'objets en un

Parfois, vous devez créer deux objets `Disposable` dans une rangée. Il existe un moyen simple d'éviter l'imbrication des blocs d' `Using` .

Ce code

```
Using File As New FileStream("MyFile", FileMode.Append)
    Using Writer As New BinaryWriter(File)
        'You code here
        Writer.Writer("Hello")
    End Using
End Using
```

peut être raccourci dans celui-ci. Le principal avantage est que vous gagnez un niveau d'indentation:

```
Using File As New FileStream("MyFile", FileMode.Append), Writer As New BinaryWriter(File)
    'You code here
    Writer.Writer("Hello")
End Using
```

Lire Objets jetables en ligne: https://riptutorial.com/fr/vb-net/topic/3204/objets-jetables

Chapitre 37: Opérateurs de court-circuit (et aussi - ou moins)

Syntaxe

- resultat = expression1 EtAlso expression2
- resultat = expression1OrElse expression2

Paramètres

Paramètre	Détails
résultat	Champs obligatoires. Toute expression booléenne. Le résultat est le résultat booléen de la comparaison des deux expressions.
expression1	Champs obligatoires. Toute expression booléenne.
expression2	Champs obligatoires. Toute expression booléenne.

Remarques

"AndAlso" et **"OrElse"** sont des opérateurs de **court** - **circuit** , ce qui signifie que l'exécution est plus courte car le compilateur n'évalue pas toutes les expressions dans une comparaison booléenne si la première fournit le résultat souhaité.

Examples

Et aussi utilisation

```
' Sometimes we don't need to evaluate all the conditions in an if statement's boolean check.

' Let's suppose we have a list of strings:

Dim MyCollection as List(Of String) = New List(of String)()

' We want to evaluate the first value inside our list:

If MyCollection.Count > 0 And MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' If MyCollection is empty, an exception will be thrown at runtime.
' This because it evaluates both first and second condition of the
' if statement regardless of the outcome of the first condition.

' Now let's apply the AndAlso operator
```

```
If MyCollection.Count > 0 AndAlso MyCollection(0).Equals("Somevalue")
    Console.WriteLine("Yes, I've found Somevalue in the collection!")
End If

' This won't throw any exception because the compiler evaluates just the first condition.
' If the first condition returns False, the second expression isn't evaluated at all.
```

Utilisation OrElse

```
' The OrElse operator is the homologous of AndAlso. It lets us perform a boolean
' comparison evaluating the second condition only if the first one is False

If testFunction(5) = True OrElse otherFunction(4) = True Then
    ' If testFunction(5) is True, otherFunction(4) is not called.
    ' Insert code to be executed.
End If
```

Éviter NullReferenceException

7.0

Ou sinon

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsOrElse(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsOr(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsOrElse(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (elements Is Nothing OrElse elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function

Public Function AverageElementsOr(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (elements Is Nothing Or elements.Count = 0) Then
        Return 0
    Else
        Return elements.Average()
    End If
End Function
```

7.0

Et aussi

```
Sub Main()
    Dim elements As List(Of Integer) = Nothing

    Dim average As Double = AverageElementsAndAlso(elements)
    Console.WriteLine(average) ' Writes 0 to Console

    Try
        'Throws ArgumentNullException
        average = AverageElementsAnd(elements)
    Catch ex As ArgumentNullException
        Console.WriteLine(ex.Message)
    End Try
End Sub

Public Function AverageElementsAndAlso(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is not called if elements is Nothing so it cannot crash
    If (Not elements Is Nothing AndAlso elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function

Public Function AverageElementsAnd(ByVal elements As IEnumerable(Of Integer)) As Double
    ' elements.Count is always called so it can crash if elements is Nothing
    If (Not elements Is Nothing And elements.Count > 0) Then
        Return elements.Average()
    Else
        Return 0
    End If
End Function
```

14.0

Visual Basic 14.0 a introduit l'opérateur conditionnel null , permettant de réécrire les fonctions de manière plus propre, en imitant le comportement de la version `AndAlso` de l'exemple.

Lire Opérateurs de court-circuit (et aussi - ou moins) en ligne: https://riptutorial.com/fr/vb-net/topic/2509/operateurs-de-court-circuit--et-aussi---ou-moins-

Chapitre 38: Option explicite

Remarques

`Option Explicit On` est une bonne pratique recommandée avec Visual Basic .Net. Il vous aide en tant que développeur à produire du code plus propre, plus stable, plus exempt de bogues et plus facile à maintenir. Dans certains cas, cela peut également vous aider à écrire des programmes plus performants!

avec ref à https://support.microsoft.com/en-in/kb/311329#bookmark-3 option strict peut également être utilisé à la place de l'option explicite. Option strict hérite de l'option explicite.

Examples

Qu'Est-ce que c'est?

Il vous oblige à déclarer explicitement toutes les variables.

Quelle est la différence entre déclarer explicitement et déclarer implicitement une variable?

Déclarer explicitement une variable:

```
Dim anInteger As Integer = 1234
```

Déclarer implicitement une variable:

```
'Did not declare aNumber using Dim
aNumber = 1234
```

Conclusion

Par conséquent, `Option Explicit On` devrait toujours être activée, car vous pourriez malpeler une variable lors de l'affectation, ce qui entraînerait un comportement inattendu de votre programme.

Comment l'allumer?

Niveau du document

Il est activé par défaut, mais vous pouvez disposer d'une couche de protection supplémentaire en plaçant `Option Explicit On` en haut du fichier de code. L'option s'appliquera à l'ensemble du document.

Niveau du projet

Vous pouvez l'activer via le menu dans Visual Studio:

Projet> Propriétés du projet> Onglet Compiler> Option Explicit

Choisissez `On` dans le menu déroulant. L'option s'appliquera à l'ensemble du document.

Tous les nouveaux projets

Vous pouvez l'activer par défaut pour tous les nouveaux projets en sélectionnant:

Outils> Options> Projets et solutions> Valeurs par défaut VB> Option Explicit

Choisissez `On` dans le menu déroulant.

Lire Option explicite en ligne: https://riptutorial.com/fr/vb-net/topic/4725/option-explicite

Chapitre 39: Option Infer

Examples

Qu'Est-ce que c'est?

Permet d'utiliser l'inférence de type local pour déclarer des variables.

Qu'est-ce que l'inférence de type?

Vous pouvez déclarer des variables locales sans indiquer explicitement un type de données. Le compilateur déduit le type de données d'une variable du type de son expression d'initialisation.

Option Infer On :

```
Dim aString  = "1234" '--> Will be treated as String by the compiler
Dim aNumber  = 4711   '--> Will be treated as Integer by the compiler
```

vs déclaration de type explicite:

```
'State a type explicitly
Dim aString as String = "1234"
Dim aNumber as Integer = 4711
```

Option Inverser Off:
Le comportement du compilateur avec `Option Infer Off` dépend du paramètre `Option Strict` qui est déjà documenté ici .

- **Option Inverser Off - Option Strict Off**
  Toutes les variables sans déclaration de type explicite sont déclarées comme `Object` .

  ```
  Dim aString  = "1234" '--> Will be treated as Object by the compiler
  ```

- **Option Infer Off - Option Strict On**
  Le compilateur ne vous laissera pas déclarer une variable sans type explicite.

  ```
  'Dim aString  = "1234" '--> Will not compile due to missing type in declaration
  ```

Comment l'activer / désactiver

Niveau du document

Il est activé par défaut, mais vous pouvez le définir en plaçant `Option Infer On|Off` en haut du fichier de code. L'option s'appliquera à l'ensemble du document.

Niveau du projet

Vous pouvez l'activer / désactiver via le menu de Visual Studio:

> Projet> Propriétés du projet> Onglet Compiler> Option infer

Choisissez `On|Off` dans le menu déroulant. L'option s'appliquera à l'ensemble du document.

Tous les nouveaux projets

Vous pouvez l'activer par défaut pour tous les nouveaux projets en sélectionnant:

> Outils> Options> Projets et solutions> Valeurs VB par défaut> Option Infer

Choisissez `On|Off` dans le menu déroulant.

Quand utiliser l'inférence de type

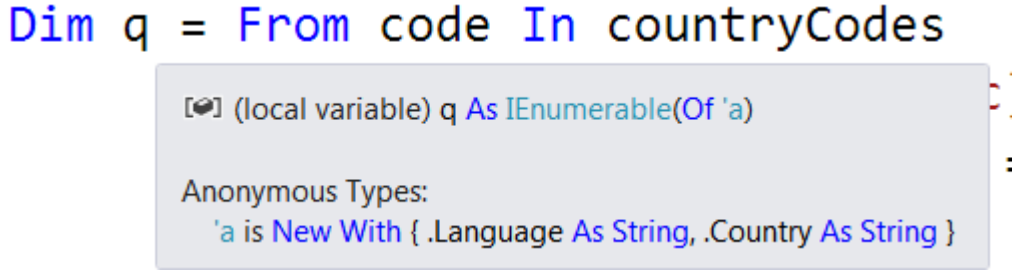
Fondamentalement, vous pouvez utiliser l'inférence de type chaque fois que cela est possible. Toutefois, soyez prudent lorsque vous `Option Infer Off` et `Option Strict Off` , car cela peut entraîner un comportement d'exécution indésirable:

```
Dim someVar = 5
someVar.GetType.ToString() '--> System.Int32
someVar = "abc"
someVar.GetType.ToString() '--> System.String
```

Type anonyme
Les types anonymes ne **peuvent** être déclarés qu'avec `Option Infer On` .
Ils sont souvent utilisés pour traiter avec LINQ :

```
Dim countryCodes = New List(Of String)
countryCodes.Add("en-US")
countryCodes.Add("en-GB")
countryCodes.Add("de-DE")
countryCodes.Add("de-AT")

Dim q = From code In countryCodes
        Let split = code.Split("-"c)
        Select New With {.Language = split(0), .Country = split(1)}
```

- **Option Infer On**
  Le compilateur reconnaîtra le type anonyme:



- **Option Infer Off**
  Le compilateur générera une erreur (avec `Option Strict On`)

ou considérera `q` comme `object` type (avec `Option Strict Off`).
Les deux cas produiront le résultat que vous ne pouvez pas utiliser le type anonyme.

Doubles / Décimales
Les variables numériques avec des décimales seront inférées comme `Double` par défaut:

```
Dim aNumber = 44.11 '--> Will be treated as type `Double` by the compiler
```

Si un autre type comme `Decimal` est souhaité, la valeur qui a initialisé la variable doit être marquée:

```
Dim mDecimal = 47.11D '--> Will be treated as type `Decimal` by the compiler
```

Lire Option Infer en ligne: https://riptutorial.com/fr/vb-net/topic/5095/option-infer

Chapitre 40: Option Strict

Syntaxe

- Option Strict {On | Off}

Remarques

`Option Strict On` est une bonne pratique recommandée avec Visual Basic .Net. Il vous aide en tant que développeur à produire du code plus propre, plus stable, plus exempt de bogues et plus facile à maintenir. Dans certains cas, cela peut également vous aider à écrire des programmes plus performants, en évitant par exemple des conversions implicites.

`On` n'est *pas* le paramètre par défaut pour une nouvelle installation de Visual Studio. Cela devrait être l'une des premières choses modifiées avant de commencer la programmation si vous allez utiliser VB.NET. La raison pour laquelle ce n'est pas le paramètre par défaut provient des premières éditions de Visual Studio lorsque les programmeurs devaient migrer des projets à partir de VB6.

Examples

Pourquoi l'utiliser?

`Option Strict On` empêche trois choses de se produire:

1. Erreurs de conversion restrictives implicites

Cela vous empêche d'attribuer à une variable *moins précise ou de plus petite capacité* (conversion *réduite*) sans conversion explicite. Cela entraînerait une perte de données.

```
Dim d As Double = 123.4
Dim s As Single = d 'This line does not compile with Option Strict On
```

2. Appels tardifs

La reliure tardive n'est pas autorisée. Cela évite les fautes de frappe qui compileraient, mais échoueraient à l'exécution

```
Dim obj As New Object
obj.Foo 'This line does not compile with Option Strict On
```

3. Erreurs de type d'objet implicite

Cela évite que la variable soit déduite comme un objet alors qu'elle aurait dû être déclarée comme un type

```
Dim something = Nothing. 'This line does not compile with Option Strict On
```

Conclusion

Sauf si vous avez besoin de faire une liaison tardive, `Option Strict On` doit toujours être `Option Strict On` car les erreurs mentionnées génèrent des erreurs de compilation au lieu d'exceptions à l'exécution.

Si vous *devez* effectuer une liaison tardive, vous pouvez *soit*

- Enveloppez tous vos appels de liaison tardifs dans une classe / module et utilisez `Option Strict Off` en haut du fichier de code (c'est la méthode préférée car cela réduit la probabilité d'une erreur de frappe dans d'autres fichiers), *ou*
- Indiquez que la liaison tardive ne provoque pas de problème de compilation (`Project Properties > Compile Tab > Warning Configuration`)

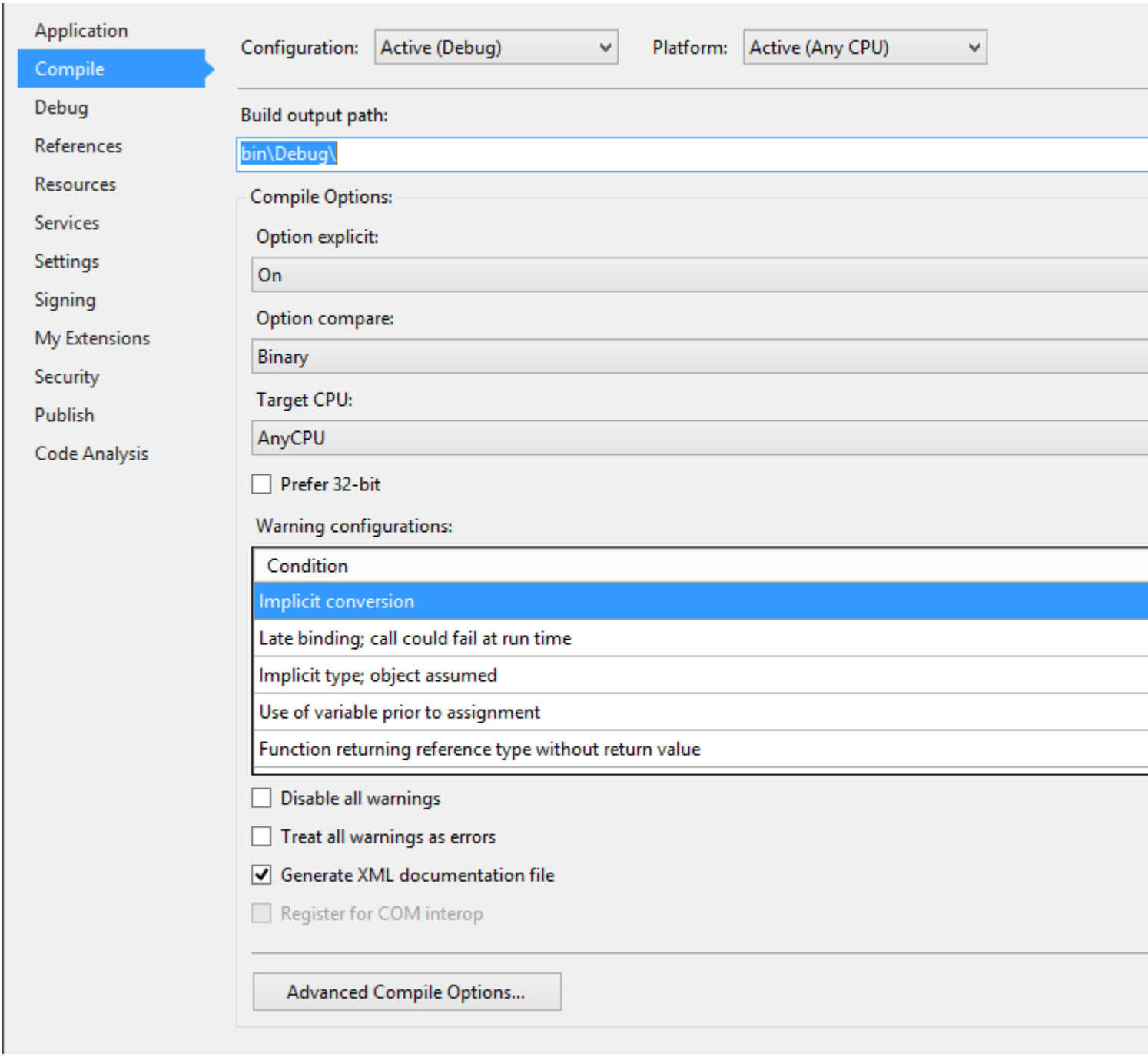
Comment l'activer

- Vous pouvez l'activer au niveau du module / classe en plaçant la directive en haut du fichier de code.

  ```
  Option Strict On
  ```

- Vous pouvez l'activer au niveau du projet via le menu de Visual Studio

  Projet> Propriétés du projet> Onglet Compiler> Option Strict> Activé



- Vous pouvez l'activer par défaut pour tous les nouveaux projets en sélectionnant:

  Outils> Options> Projets et solutions> Valeurs par défaut VB> Option Strict
  Réglez-le sur `On` .

Lire Option Strict en ligne: https://riptutorial.com/fr/vb-net/topic/4022/option-strict

Chapitre 41: Récursivité

Examples

Calculez le nième nombre de Fibonacci

Visuel Basic.NET, comme la plupart des langues, permet récursion, un processus par lequel une fonction *elle - même* appelle dans certaines conditions.

Voici une fonction de base dans Visual Basic .NET pour calculer les nombres de Fibonacci .

```
''' <summary>
''' Gets the n'th Fibonacci number
''' </summary>
''' <param name="n">The 1-indexed ordinal number of the Fibonacci sequence that you wish to
receive. Precondition: Must be greater than or equal to 1.</param>
''' <returns>The nth Fibonacci number. Throws an exception if a precondition is
violated.</returns>
Public Shared Function Fibonacci(ByVal n as Integer) as Integer
    If n<1
        Throw New ArgumentOutOfRangeException("n must be greater than or equal to one.")
    End If
    If (n=1) or (n=2)
        ''Base case. The first two Fibonacci numbers (n=1 and n=2) are both 1, by definition.
        Return 1
    End If
    ''Recursive case.
    ''Get the two previous Fibonacci numbers via recursion, add them together, and return the
result.
    Return Fibonacci(n-1) + Fibonacci(n-2)
End Function
```

Cette fonction vérifie d'abord si la fonction a été appelée avec le paramètre `n` égal à `1` ou `2` . Par définition, les deux premières valeurs de la séquence de Fibonacci sont 1 et 1, donc aucun calcul supplémentaire n'est nécessaire pour le déterminer. Si `n` est supérieur à 2, nous ne pouvons pas rechercher la valeur associée aussi facilement, mais nous savons qu'un tel nombre de Fibonacci est égal à la somme des deux nombres précédents. Nous les demandons donc par *récursivité* (appelant notre propre fonction Fibonacci). Comme les appels récursifs successifs sont appelés avec des nombres de plus en plus petits via des décréments de -1 et -2, nous savons qu'ils finiront par atteindre des nombres inférieurs à 2. Une fois que ces conditions (appelées *cas de base*) sont atteintes, la pile se déroule obtenir notre résultat final

Lire Récursivité en ligne: https://riptutorial.com/fr/vb-net/topic/7862/recursivite

Chapitre 42: Réflexion

Examples

Récupérer des propriétés pour une instance d'une classe

```
Imports System.Reflection

Public Class PropertyExample

    Public Function GetMyProperties() As PropertyInfo()
        Dim objProperties As PropertyInfo()
        objProperties = Me.GetType.GetProperties(BindingFlags.Public Or BindingFlags.Instance)
        Return objProperties
    End Function

    Public Property ThisWillBeRetrieved As String = "ThisWillBeRetrieved"

    Private Property ThisWillNot As String = "ThisWillNot"

    Public Shared Property NeitherWillThis As String = "NeitherWillThis"

    Public Overrides Function ToString() As String
        Return String.Join(",", GetMyProperties.Select(Function(pi) pi.Name).ToArray)
    End Function
End Class
```

Le paramètre de GetProperties définit les types de propriétés qui seront renvoyés par la fonction. Puisque nous passons Public et Instance, la méthode ne renverra que les propriétés à la fois publiques et non partagées. Reportez-vous à la section Attribut Drapeaux et explication sur la manière de combiner Flag-enums.

Obtenir les membres d'un type

```
Dim flags = BindingFlags.Static Or BindingFlags.Public Or BindingFlags.Instance
Dim members = GetType(String).GetMembers(flags)
For Each member In members
    Console.WriteLine($"{member.Name}, ({member.MemberType})")
Next
```

Obtenez une méthode et invoquez-la

Méthode statique:

```
Dim parseMethod = GetType(Integer).GetMethod("Parse",{GetType(String)})
Dim result = DirectCast(parseMethod.Invoke(Nothing,{"123"}), Integer)
```

Méthode d'instance:

```
 Dim instance = "hello".ToUpper
```

```
Dim method = Gettype(String).GetMethod("ToUpper",{})
Dim result = method.Invoke(instance,{})
Console.WriteLine(result) 'HELLO
```

Créer une instance d'un type générique

```
    Dim openListType = GetType(List(Of ))
    Dim typeParameters = {GetType(String)}
    Dim stringListType = openListType.MakeGenericType(typeParameters)
    Dim instance = DirectCast(Activator.CreateInstance(stringListType), List(Of String))
    instance.Add("Hello")
```

Lire Réflexion en ligne: https://riptutorial.com/fr/vb-net/topic/1598/reflexion

Chapitre 43: Rendez-vous amoureux

Examples

Conversion (analyse) d'une chaîne en une date

Si vous connaissez le format de la chaîne que vous convertissez (analyse), vous devez utiliser `DateTime.ParseExact`

```
Dim dateString As String = "12.07.2003"
Dim dateFormat As String = "dd.MM.yyyy"
Dim dateValue As Date

dateValue = DateTime.ParseExact(dateString, dateFormat,
Globalization.CultureInfo.InvariantCulture)
```

Si vous n'êtes pas certain du format de la chaîne, vous pouvez utiliser `DateTime.TryParseExact` et tester le résultat pour voir s'il est analysé ou non:

```
Dim dateString As String = "23-09-2013"
Dim dateFormat As String = "dd-MM-yyyy"
Dim dateValue As Date

If DateTime.TryParseExact(dateString, dateFormat, Globalization.CultureInfo.InvariantCulture,
DateTimeStyles.None, dateValue) Then
    'the parse worked and the dateValue variable now holds the datetime that was parsed as it
is passing in ByRef
Else
    'the parse failed
End If
```

Conversion d'une date en chaîne

Utilisez simplement la surcharge `.ToString` d'un objet `DateTime` pour obtenir le format requis:

```
Dim dateValue As DateTime = New DateTime(2001, 03, 06)
Dim dateString As String = dateValue.ToString("yyyy-MM-dd") '2001-03-06
```

Lire Rendez-vous amoureux en ligne: https://riptutorial.com/fr/vb-net/topic/3727/rendez-vous-amoureux

Chapitre 44: Serveur ftp

Syntaxe

- My.Computer.Network.DownloadFile (serverFile As String, localFile As String)
- My.Computer.Network.DownloadFile (serverFile As String, localFile As String, utilisateur As String, mot de passe As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String)
- My.Computer.Network.UploadFile (localFile As String, serverFile As String, utilisateur As String, mot de passe As String)

Examples

Télécharger le fichier depuis le serveur FTP

```
My.Computer.Network.DownloadFile("ftp://server.my/myfile.txt", "donwloaded_file.txt")
```

Cette commande télécharge le fichier `myfile.txt` du serveur nommé `server.my` et l'enregistre sous le nom `donwloaded_file.txt` dans le répertoire de travail. Vous pouvez spécifier un chemin absolu pour le fichier téléchargé.

Télécharger le fichier du serveur FTP lorsque la connexion est requise

```
My.Computer.Network.DownloadFile("ftp://srv.my/myfile.txt", "donwload.txt", "Peter", "1234")
```

Cette commande télécharge le fichier `myfile.txt` du serveur nommé `srv.my` et l'enregistre sous le nom `donwload.txt` dans le répertoire de travail. Vous pouvez spécifier un chemin absolu pour le fichier téléchargé. Le fichier est téléchargé par l'utilisateur Peter avec le mot de passe 1234.

Télécharger le fichier sur le serveur FTP

```
My.Computer.Network.UploadFile("example.txt", "ftp://server.my/server_example.txt")
```

Cette commande télécharge le fichier `example.txt` répertoire de travail (vous pouvez spécifier un chemin absolu si vous le souhaitez) vers le serveur nommé `server.my` . Le fichier stocké sur le serveur sera nommé `server_example.txt` .

Télécharger le fichier sur le serveur FTP lorsque la connexion est requise

```
My.Computer.Network.UploadFile("doc.txt", "ftp://server.my/on_server.txt", "Peter", "1234")
```

Cette commande télécharge le fichier `doc.txt` répertoire de travail (vous pouvez spécifier un chemin absolu si vous le souhaitez) vers le serveur nommé `server.my` . Le fichier stocké sur le

serveur sera nommé `server_example.txt` . Fill est envoyé sur le serveur par l'utilisateur Peter et le mot de passe 1234.

Lire Serveur ftp en ligne: https://riptutorial.com/fr/vb-net/topic/4078/serveur-ftp

Chapitre 45: Tableau

Remarques

```
Dim myArray(2) As Integer

someFunc(myArray)
```

Un tableau est une collection d'objets classés par index. Le type d'objet est défini par le type indiqué dans la déclaration de tableau.

Les tableaux dans Visual Basic .NET sont le plus souvent (et par défaut) zéro (0), ce qui signifie que le premier index est 0. Un tableau de 10 éléments aura une plage d'index de 0 à 9. Lors de l'accès à des éléments de tableau, l'index maximal accessible est inférieur à celui du nombre total d'éléments. De ce fait, les boucles qui accèdent de manière incrémentielle aux index de tableau doivent toujours effectuer une vérification de plage où la valeur est inférieure à la longueur du tableau.

Examples

Définition de tableau

```
Dim array(9) As Integer ' Defines an array variable with 10 Integer elements (0-9).

Dim array = New Integer(10) {} ' Defines an array variable with 11 Integer elements (0-10)
                               'using New.

Dim array As Integer() = {1, 2, 3, 4} ' Defines an Integer array variable and populate it
                                      'using an array literal. Populates the array with
                                      '4 elements.

ReDim Preserve array(10) ' Redefines the size of an existing array variable preserving any
                         'existing values in the array. The array will now have 11 Integer
                         'elements (0-10).

ReDim array(10) ' Redefines the size of an existing array variable discarding any
                'existing values in the array. The array will now have 11 Integer
                'elements (0-10).
```

Base zéro

Tous les tableaux de VB.NET sont basés sur zéro. En d'autres termes, l'index du premier élément (la limite inférieure) dans un tableau VB.NET est toujours `0` . Les anciennes versions de VB, telles que VB6 et VBA, étaient basées sur une base par défaut, mais elles permettaient de remplacer les limites par défaut. Dans les versions antérieures de VB, les limites inférieure et supérieure pouvaient être explicitement définies (par exemple, `Dim array(5 To 10)` . Dans VB.NET, afin de maintenir la compatibilité avec les autres langages .NET, cette flexibilité était supprimée et la

limite inférieure de `0` est maintenant toujours appliqué.Toutefois, la syntaxe `To` peut toujours être utilisée dans VB.NET, ce qui peut rendre la plage plus claire: par exemple, les exemples suivants sont tous équivalents à ceux énumérés ci-dessus:

```
Dim array(0 To 9) As Integer

Dim array = New Integer(0 To 10) {}

ReDim Preserve array(0 To 10)

ReDim array(0 To 10)
```

Déclarations de tableaux imbriqués

```
Dim myArray = {{1, 2}, {3, 4}}
```

Déclarez un tableau à une seule dimension et définissez les valeurs des éléments du tableau

```
Dim array = New Integer() {1, 2, 3, 4}
```

ou

```
Dim array As Int32() = {1, 2, 3, 4}
```

Initialisation du tableau

```
Dim array() As Integer = {2, 0, 1, 6}                   ''Initialize an array of four
Integers.
Dim strings() As String = {"this", "is", "an", "array"} ''Initialize an array of four Strings.
Dim floats() As Single = {56.2, 55.633, 1.2, 5.7743, 22.345}
              ''Initialize an array of five Singles, which are the same as floats in C#.
Dim miscellaneous() as Object = { New Object(), "Hello", New List(of String) }
              ''Initialize an array of three references to any reference type objects
              ''and point them to objects of three different types.
```

Initialisation du tableau multidimensionnel

```
Dim array2D(,) As Integer = {{1, 2, 3}, {4, 5, 6}}
' array2D(0, 0) is 1 ; array2D(0, 1) is 2 ; array2D(1, 0) is 4

Dim array3D(,,) As Integer = {{{1, 2, 3}, {4, 5, 6}}, {{7, 8, 9}, {10, 11, 12}}}
' array3D(0, 0, 0) is 1 ; array3D(0, 0, 1) is 2
' array3D(0, 1, 0) is 4 ; array3D(1, 0, 0) is 7
```

Initialisation des tableaux dentelés

Notez la parenthèse pour faire la distinction entre un tableau irrégulier et un tableau multidimensionnel. SubArrays peut être de longueur différente

```
Dim jaggedArray()() As Integer = { ({1, 2, 3}), ({4, 5, 6}), ({7}) }
' jaggedArray(0) is {1, 2, 3} and so jaggedArray(0)(0) is 1
' jaggedArray(1) is {4, 5, 6} and so jaggedArray(1)(0) is 4
' jaggedArray(2) is {7} and so jaggedArray(2)(0) is 7
```

Variables de tableau nul

Les tableaux étant des types de référence, une variable de tableau peut être nulle. Pour déclarer une variable de tableau nul, vous devez la déclarer sans taille:

```
Dim array() As Integer
```

Ou

```
Dim array As Integer()
```

Pour vérifier si un tableau est nul, testez pour voir s'il `Is Nothing` :

```
Dim array() As Integer
If array Is Nothing Then
    array = {1, 2, 3}
End If
```

Pour définir une variable de tableau existante sur null, définissez-la simplement sur `Nothing` :

```
Dim array() As Integer = {1, 2, 3}
array = Nothing
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Ou utilisez `Erase` , qui fait la même chose:

```
Dim array() As Integer = {1, 2, 3}
Erase array
Console.WriteLine(array(0))  ' Throws a NullReferenceException
```

Référencement du même tableau à partir de deux variables

Les tableaux étant des types de référence, il est possible d'avoir plusieurs variables pointant vers le même objet tableau.

```
Dim array1() As Integer = {1, 2, 3}
Dim array2() As Integer = array1
array1(0) = 4
Console.WriteLine(String.Join(", ", array2))  ' Writes "4, 2, 3"
```

Limites inférieures non nulles

Avec `Option Strict On` , bien que le .NET Framework permette la création de tableaux à dimension unique avec des limites inférieures à zéro, ils ne sont pas des "vecteurs" et ne sont donc pas

compatibles avec les tableaux typés VB.NET. Cela signifie qu'ils ne peuvent être vus que sous forme de `Array` et ne peuvent donc pas utiliser des références de tableau `(index)` normales.

```
Dim a As Array = Array.CreateInstance(GetType(Integer), {4}, {-1})
For y = LBound(a) To UBound(a)
    a.SetValue(y * y, y)
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {a.GetValue(y)}")
Next
```

De même qu'en utilisant `Option Strict Off` , vous pouvez récupérer la syntaxe `(index)` en traitant le tableau comme un `IList` , mais ce n'est pas un tableau, vous ne pouvez donc pas utiliser `LBound` et `UBound` sur ce nom de variable (et vous) ne pas toujours éviter la boxe):

```
Dim nsz As IList = a
For y = LBound(a) To UBound(a)
    nsz(y) = 2 - CInt(nsz(y))
Next
For y = LBound(a) To UBound(a)
    Console.WriteLine($"{y}: {nsz(y)}")
Next
```

Les tableaux de limites inférieures multidimensionnelles non nulles *sont* compatibles avec les tableaux typés multidimensionnels VB.NET:

```
Dim nza(,) As Integer = DirectCast(Array.CreateInstance(GetType(Integer),
                                    {4, 3}, {1, -1}), Integer(,))
For y = LBound(nza) To UBound(nza)
    For w = LBound(nza, 2) To UBound(nza, 2)
        nza(y, w) = -y * w + nza(UBound(nza) - y + LBound(nza),
                                 UBound(nza, 2) - w + LBound(nza, 2))
    Next
Next
For y = LBound(nza) To UBound(nza)
    Dim ly = y
    Console.WriteLine(String.Join(" ",
        Enumerable.Repeat(ly & ":", 1).Concat(
            Enumerable.Range(LBound(nza, 2), UBound(nza, 2) - LBound(nza, 2) + 1) _
            .Select(Function(w) CStr(nza(ly, w))))))
Next
```

Référence MSDN: Array.CreateInstance

Lire Tableau en ligne: https://riptutorial.com/fr/vb-net/topic/805/tableau

Chapitre 46: Test d'unité dans VB.NET

Remarques

Ceci est l'exemple le plus simple mais descriptif pour la procédure de test unitaire. N'hésitez pas à ajouter d'autres méthodes pour vérifier les différents types de données.

Examples

Test unitaire pour le calcul de la taxe

Cet exemple est divisé en deux piliers

- **Classe de** calcul de salaire: Calcul du salaire net après déduction fiscale
- **Classe SalaryCalculationTests** : Pour tester la méthode qui calcule le salaire net

Étape 1: Créer une bibliothèque de classes, nommez-la **WagesLibrary** ou un nom approprié. Puis renommez la classe en **SalaryCalculation**

" " " Class for Salary Calculations " 'Classe publique SalaireCalculation

```
    ''' <summary>
    ''' Employee Salary
    ''' </summary>
    Public Shared Salary As Double

    ''' <summary>
    ''' Tax fraction (0-1)
    ''' </summary>
    Public Shared Tax As Double

    ''' <summary>
    ''' Function to calculate Net Salary
    ''' </summary>
    ''' <returns></returns>
    Public Shared Function CalculateNetSalary()
        Return Salary - Salary * Tax
    End Function
End Class
```

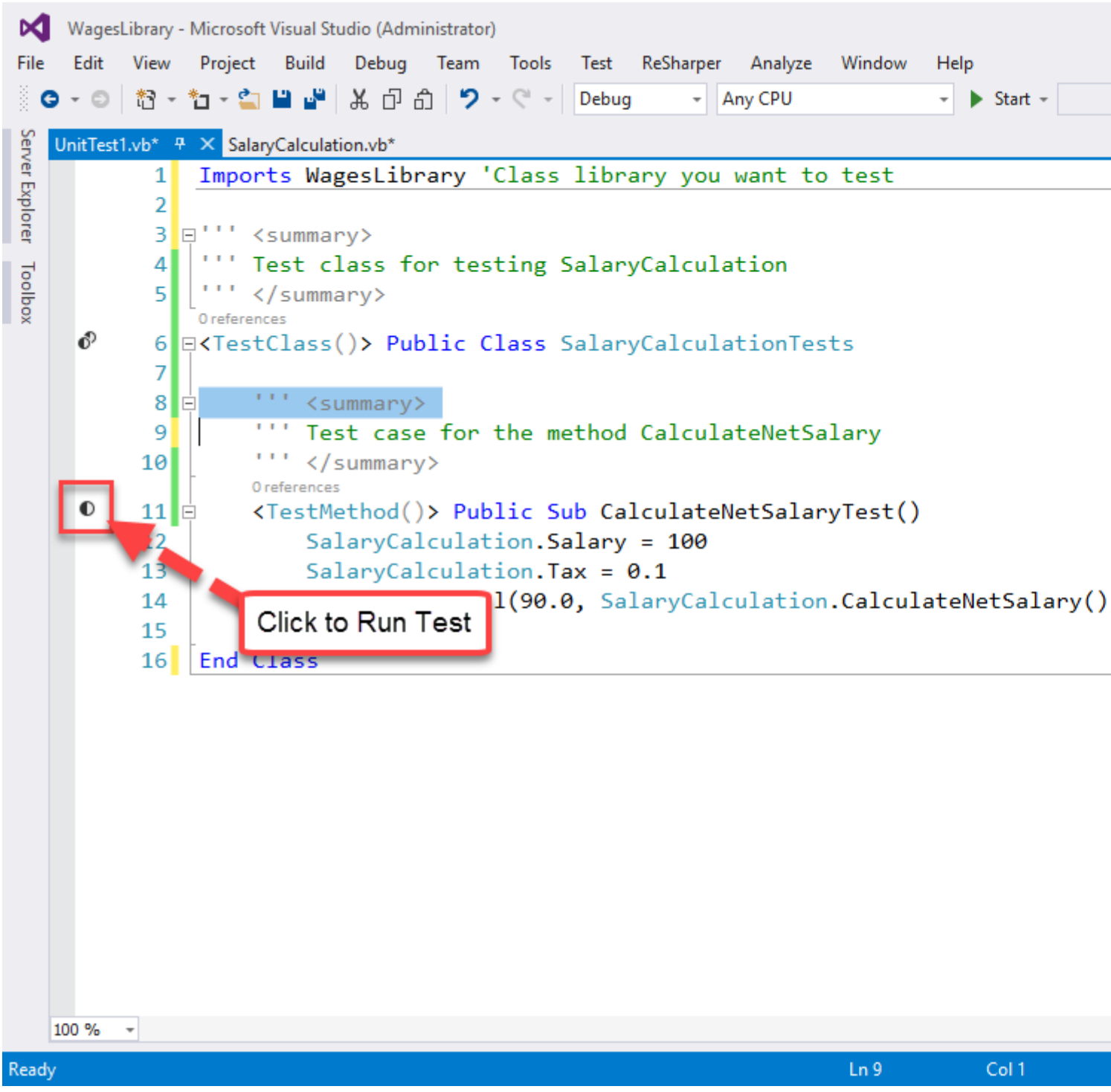
Étape 2 : Créer un projet de test unitaire. Ajouter une référence à la bibliothèque de classes créée et coller le code ci-dessous

```
Imports WagesLibrary 'Class library you want to test

''' <summary>
''' Test class for testing SalaryCalculation
''' </summary>
<TestClass()> Public Class SalaryCalculationTests

    ''' <summary>
```

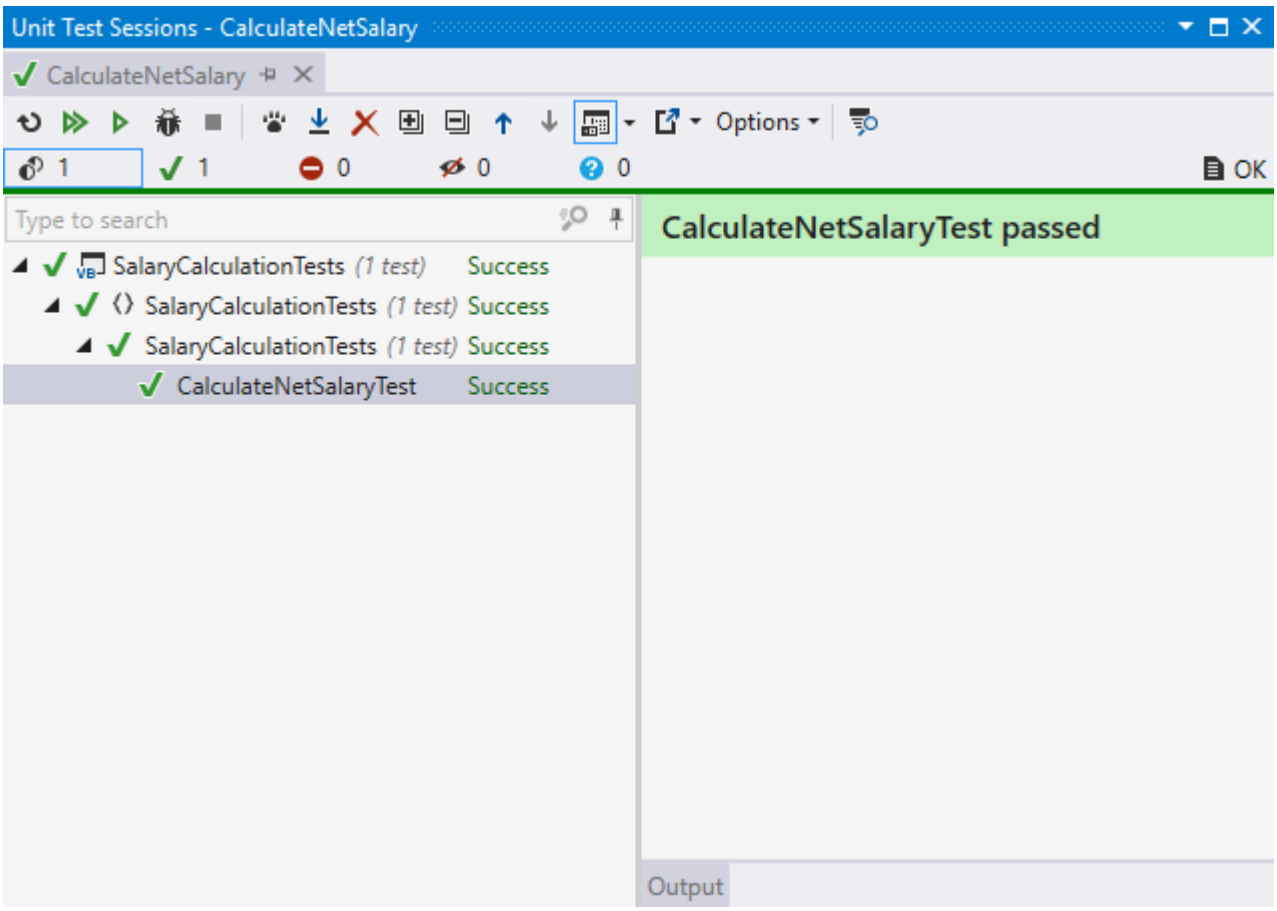
```
    ''' Test case for the method CalculateNetSalary
    ''' </summary>
    <TestMethod()> Public Sub CalculateNetSalaryTest()
        SalaryCalculation.Salary = 100
        SalaryCalculation.Tax = 0.1
        Assert.AreEqual(90.0, SalaryCalculation.CalculateNetSalary(), 0.1)
    End Sub
End Class
```

`Assert.Equal` vérifie la valeur attendue par rapport à la valeur calculée réelle. la valeur `0.1` est utilisée pour permettre la tolérance ou la variation entre *le* résultat *attendu* et *le* résultat *réel* .

Étape 3: Exécuter le test de la méthode pour voir le résultat



Résultat du test



Test de la classe d'employé affecté et dérivé

Cet exemple a plus de tests disponibles dans les tests unitaires.

Employee.vb (bibliothèque de classes)

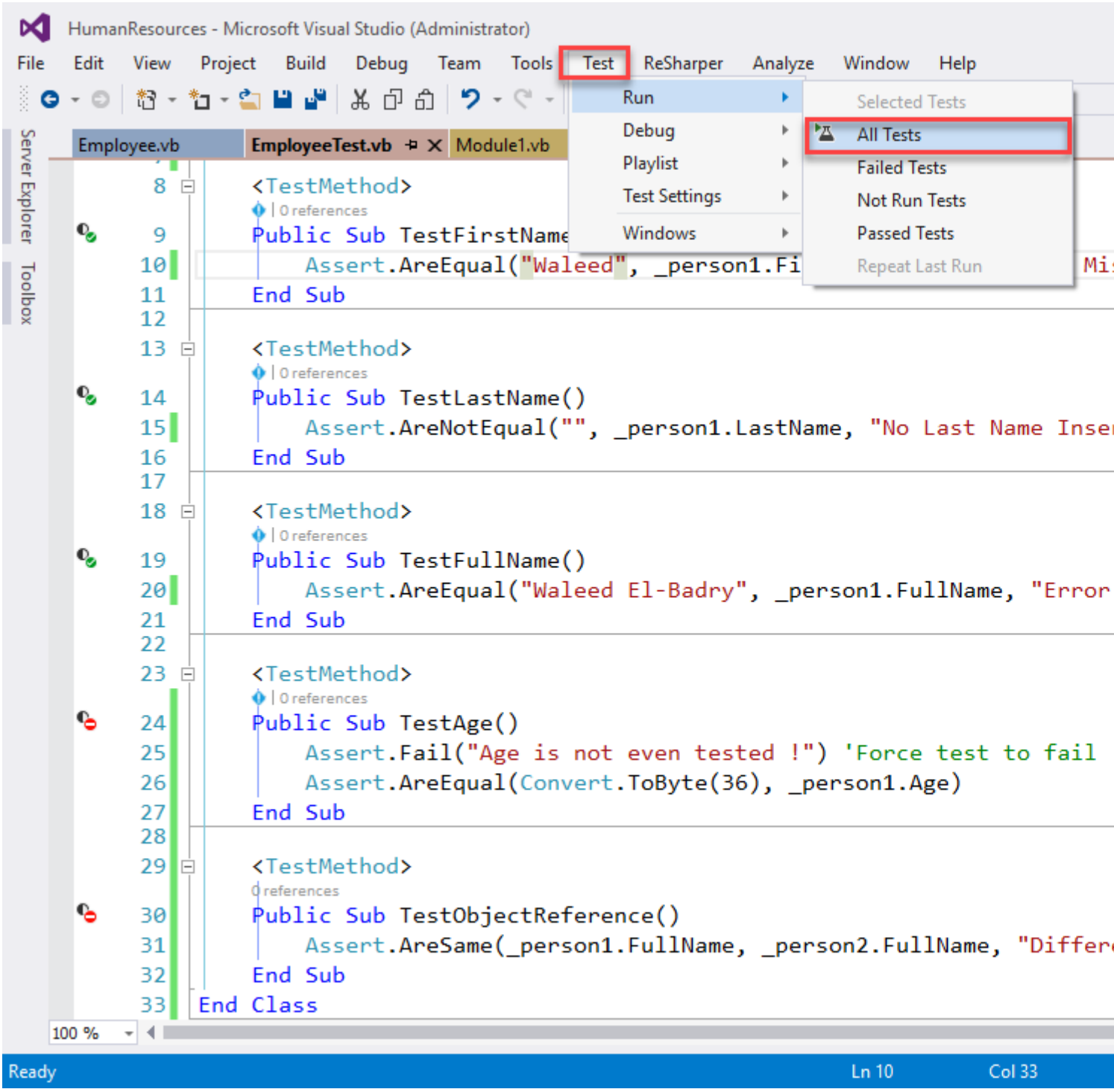
```
''' <summary>
''' Employee Class
''' </summary>
Public Class Employee

    ''' <summary>
    ''' First name of employee
    ''' </summary>
    Public Property FirstName As String = ""

    ''' <summary>
    ''' Last name of employee
    ''' </summary>
    Public Property LastName As String = ""

    ''' <summary>
    ''' Full name of employee
    ''' </summary>
    Public ReadOnly Property FullName As String = ""

    ''' <summary>
    ''' Employee's age
    ''' </summary>
```

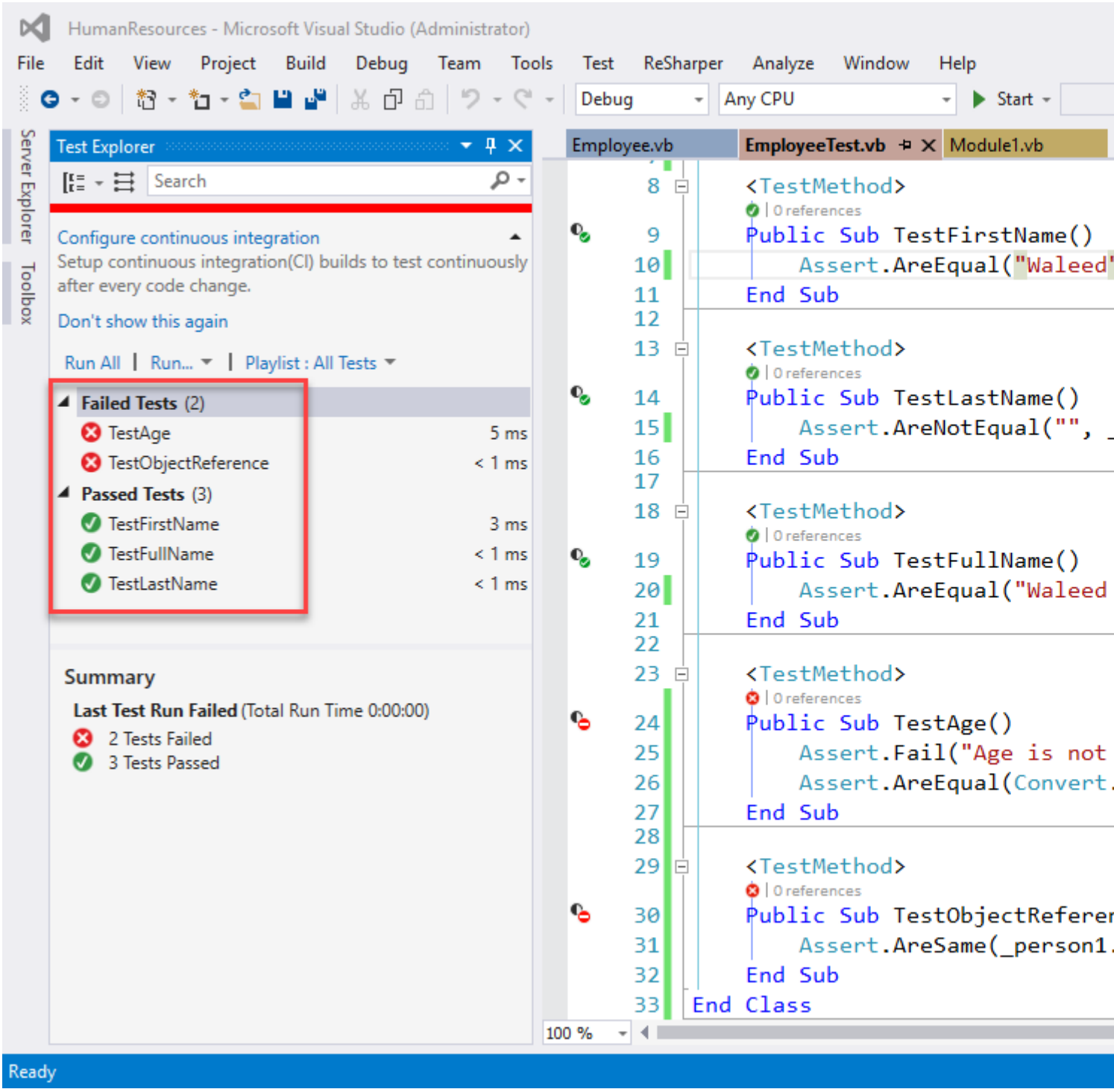
```
    Public Property Age As Byte

    ''' <summary>
    ''' Instantiate new instance of employee
    ''' </summary>
    ''' <param name="firstName">Employee first name</param>
    ''' <param name="lastName">Employee last name</param>
    Public Sub New(firstName As String, lastName As String, dateofbirth As Date)
        Me.FirstName = firstName
        Me.LastName = lastName
        FullName = Me.FirstName + " " + Me.LastName
        Age = Convert.ToByte(Date.Now.Year - dateofbirth.Year)
    End Sub
End Class
```

EmployeeTest.vb (projet de test)

```
Imports HumanResources

<TestClass()>
Public Class EmployeeTests
    ReadOnly _person1 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))
    ReadOnly _person2 As New Employee("Waleed", "El-Badry", New DateTime(1980, 8, 22))

    <TestMethod>
    Public Sub TestFirstName()
        Assert.AreEqual("Waleed", _person1.FirstName, "First Name Mismatch")
    End Sub

    <TestMethod>
    Public Sub TestLastName()
        Assert.AreNotEqual("", _person1.LastName, "No Last Name Inserted!")
    End Sub

    <TestMethod>
    Public Sub TestFullName()
        Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error in concatination of
names")
    End Sub

    <TestMethod>
    Public Sub TestAge()
        Assert.Fail("Age is not even tested !") 'Force test to fail !
        Assert.AreEqual(Convert.ToByte(36), _person1.Age)
    End Sub

    <TestMethod>
    Public Sub TestObjectReference()
        Assert.AreSame(_person1.FullName, _person2.FullName, "Different objects with same
data")
    End Sub
End Class
```

Résultat après exécution des tests


HumanResources - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Tools Test ReSharper Analyze Window Help
Run
Debug
Playlist
Test Settings
Windows
Selected Tests
All Tests
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run
Server Explorer
Toolbox
Employee.vb
EmployeeTest.vb
Module1.vb
<TestMethod>
0 references
Public Sub TestFirstName
Assert.AreEqual("Waleed", _person1.Fi
End Sub
<TestMethod>
0 references
Public Sub TestLastName()
Assert.AreNotEqual("", _person1.LastName, "No Last Name Inse
End Sub
<TestMethod>
0 references
Public Sub TestFullName()
Assert.AreEqual("Waleed El-Badry", _person1.FullName, "Error
End Sub
<TestMethod>
0 references
Public Sub TestAge()
Assert.Fail("Age is not even tested !") 'Force test to fail
Assert.AreEqual(Convert.ToByte(36), _person1.Age)
End Sub
<TestMethod>
0 references
Public Sub TestObjectReference()
Assert.AreSame(_person1.FullName, _person2.FullName, "Differ
End Sub
End Class
100 %
Ready
Ln 10
Col 33



Lire Test d'unité dans VB.NET en ligne: https://riptutorial.com/fr/vb-net/topic/6843/test-d-unite-dans-vb-net

Chapitre 47: Travailler avec Windows Forms

Examples

En utilisant l'instance de formulaire par défaut

VB.NET offre des instances de formulaire par défaut. Le développeur n'a pas besoin de créer l'instance telle qu'elle est créée en arrière-plan. Cependant, *il n'est **pas** préférable* d'utiliser l'instance par défaut sauf les programmes les plus simples.

```
Public Class Form1

    Public Sub Foo()
        MessageBox.Show("Bar")
    End Sub

End Class

Module Module1

    Public Sub Main()
        ' Default instance
        Form1.Foo()
        ' New instance
        Dim myForm1 As Form1 = New Form1()
        myForm1.Foo()

    End Sub

End Module
```

Voir également:

- Devez-vous explicitement créer une instance de formulaire dans VB.NET?
- Pourquoi existe-t-il une instance par défaut de chaque formulaire dans VB.Net mais pas dans C #?

Transmission de données d'un formulaire à un autre

Parfois, vous souhaiterez peut-être transmettre des informations générées sous une forme à une autre pour une utilisation supplémentaire. Ceci est utile pour les formulaires qui affichent un outil de recherche ou une page de paramètres parmi de nombreuses autres utilisations.

Supposons que vous souhaitiez passer un `DataTable` entre un formulaire déjà ouvert *(MainForm)* et un nouveau formulaire *(NewForm)* :

Dans le MainForm:

```
  Private Sub Open_New_Form()
       Dim NewInstanceOfForm As New NewForm(DataTable1)
```

```
        NewInstanceOfForm.ShowDialog()
    End Sub
```

Dans le NewForm

```
Public Class NewForm
    Dim NewDataTable as Datatable

    Public Sub New(PassedDataTable As Datatable)
        InitializeComponent()
        NewDataTable= PassedDataTable
    End Sub

End Class
```

Maintenant, quand le *NewForm* est ouvert, il est passé `DataTable1` de *MainForm* et stocké comme `NewDataTable` dans *NewForm* pour être utilisé par ce formulaire.

Cela peut être extrêmement utile lorsque vous tentez de transmettre de grandes quantités d'informations entre les formulaires, en particulier lorsque vous combinez toutes les informations dans une seule `ArrayList` et que vous transmettez `ArrayList` au nouveau formulaire.

Lire Travailler avec Windows Forms en ligne: https://riptutorial.com/fr/vb-net/topic/4636/travailler-avec-windows-forms

Chapitre 48: Utiliser axWindowsMediaPlayer dans VB.Net

Introduction

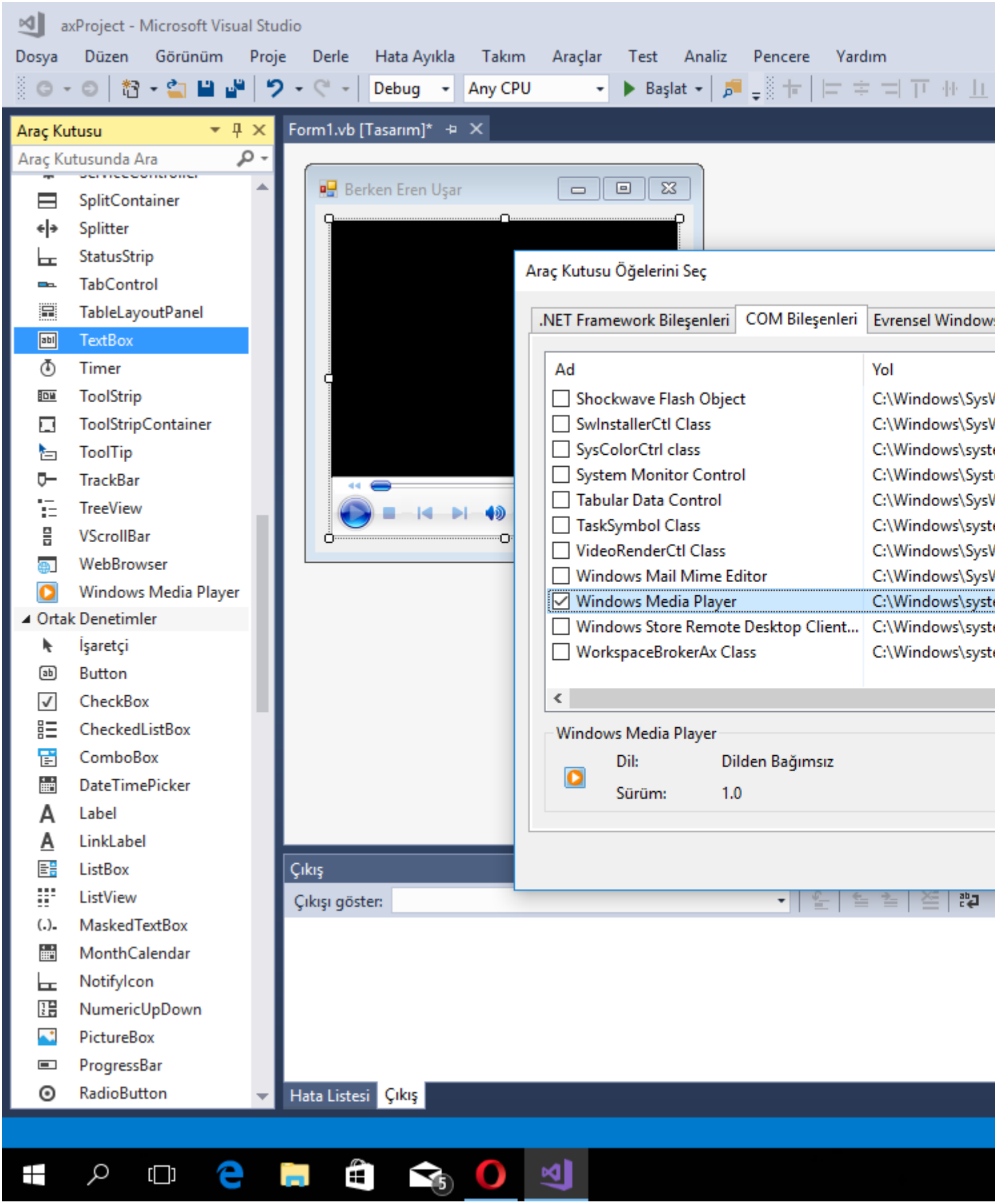
axWindowsMediaPlayer est le contrôle des fichiers multimédia tels que les vidéos et la musique.

Examples

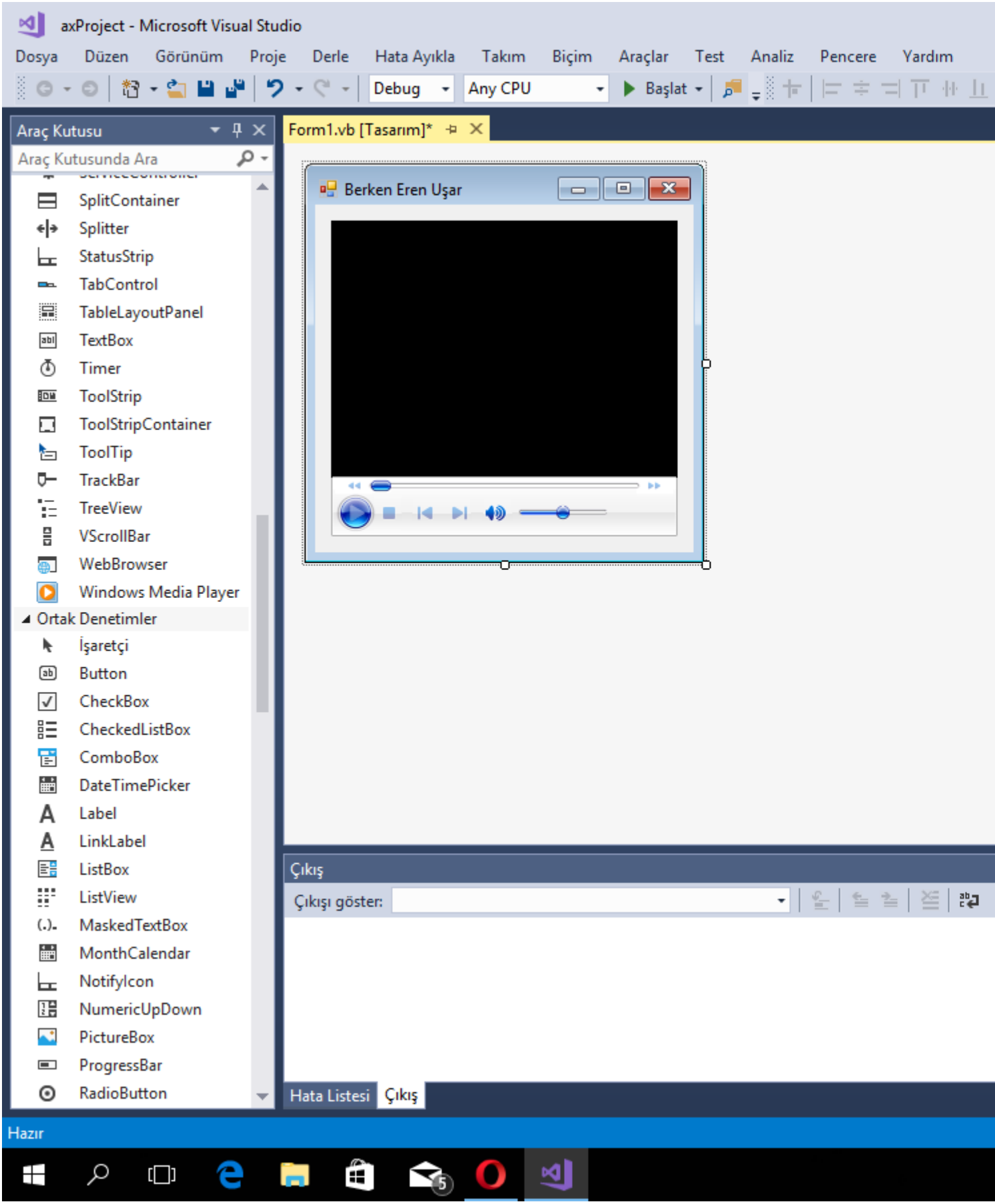
Ajouter l'axWindowsMediaPlayer

- Cliquez avec le bouton droit sur la boîte à outils, puis cliquez sur "Choisir les éléments".
- Sélectionnez l'onglet Composants COM, puis vérifiez sur le Lecteur Windows Media.
- axWindowsMediaPlayer sera ajouté à Toolbox.

Cochez cette case pour utiliser axWindowsMediaPlayer



Ensuite, vous pouvez utiliser axWindowsMediaPlayer :)



Jouer un fichier multimédia

```
AxWindowsMediaPlayer1.URL = "C:\My Files\Movies\Avatar.mp4"
```

```
AxWindowsMediaPlayer1.Ctlcontrols.play()
```

Ce code jouera Avatar dans le axWindowsMediaPlayer.

Lire Utiliser axWindowsMediaPlayer dans VB.Net en ligne: https://riptutorial.com/fr/vb-net/topic/10096/utiliser-axwindowsmediaplayer-dans-vb-net

Chapitre 49: Utiliser BackgroundWorker

Examples

Implémentation de base de la classe de travail d'arrière-plan

Vous devez importer System.ComponentModel pour utiliser le travailleur d'arrière-plan

```
Imports System.ComponentModel
```

Puis déclarer une variable privée

```
Private bgWorker As New BackgroundWorker
```

Vous devez créer deux méthodes pour les événements DoWork et RunWorkerCompleted du travail en arrière-plan et les affecter.

```
Private Sub MyWorker_DoWork(ByVal sender As System.Object, ByVal e As
System.ComponentModel.DoWorkEventArgs)
  'Add your codes here for the worker to execute

End Sub
```

Le sous ci-dessous sera exécuté lorsque le travailleur termine le travail

```
Private Sub MyWorker_RunWorkerCompleted(ByVal sender As Object, ByVal e As
System.ComponentModel.RunWorkerCompletedEventArgs)
    'Add your codes for the worker to execute after finishing the work.

End Sub
```

Ensuite, dans votre code, ajoutez les lignes ci-dessous pour démarrer le travailleur d'arrière-plan

```
    bgWorker = New BackgroundWorker
    AddHandler bgWorker.DoWork, AddressOf MyWorker_DoWork
    AddHandler bgWorker.RunWorkerCompleted, AddressOf MyWorker_RunWorkerCompleted
    bgWorker.RunWorkerAsync()
```

Lorsque vous appelez la fonction RunWorkerAsync (), MyWorker_DoWork sera exécuté.

Lire Utiliser BackgroundWorker en ligne: https://riptutorial.com/fr/vb-net/topic/6401/utiliser-backgroundworker

Chapitre 50: Utiliser la déclaration

Syntaxe

- Utilisation de *a* = New *DisposableClass* [, *b* = ...]
  ...
  Terminer en utilisant
- Utiliser *a* = *GetDisposable (...)* [, *b* = ...]
  ...
  Terminer en utilisant

Examples

Voir les exemples sous Objets jetables

Concept de base d'IDisposable

Lire Utiliser la déclaration en ligne: https://riptutorial.com/fr/vb-net/topic/7965/utiliser-la-declaration

Chapitre 51: WinForms SpellCheckBox

Introduction

Exemple sur la façon d'ajouter une case à cocher orthographique à une application WindowsForms. Cet exemple ne nécessite pas l'installation de Word, ni l'utilisation de Word.

Il utilise Interop WPF en utilisant le contrôle ElementHost pour créer un UserControl WPF à partir d'un TextBox WPF. WPF TextBox a une fonction intégrée pour la vérification orthographique. Nous allons exploiter cette fonction intégrée plutôt que de compter sur un programme externe.

Examples

ElementHost WPF TextBox

Cet exemple a été modélisé à partir d'un exemple trouvé sur Internet. Je ne peux pas trouver le lien ou je donnerais le crédit à l'auteur. J'ai pris l'échantillon que j'ai trouvé et l'ai modifié pour fonctionner pour mon application.

1. Ajoutez les références suivantes:

   System.Xaml, PresentationCore, PresentationFramework, WindowsBase et WindowsFormsIntegration

2. Créer une nouvelle classe et passé ce code

```
Imports System
Imports System.ComponentModel
Imports System.ComponentModel.Design.Serialization
Imports System.Windows
Imports System.Windows.Controls
Imports System.Windows.Forms.Integration
Imports System.Windows.Forms.Design

<Designer(GetType(ControlDesigner))> _
Class SpellCheckBox
Inherits ElementHost

Private box As TextBox

Public Sub New()
    box = New TextBox()
    MyBase.Child = box
    AddHandler box.TextChanged, AddressOf box_TextChanged
    box.SpellCheck.IsEnabled = True
    box.VerticalScrollBarVisibility = ScrollBarVisibility.Auto
    Me.Size = New System.Drawing.Size(100, 20)
End Sub

Private Sub box_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
    OnTextChanged(EventArgs.Empty)
```

```
End Sub

<DefaultValue("")> _
Public Overrides Property Text() As String
    Get
        Return box.Text
    End Get
    Set(ByVal value As String)
        box.Text = value
    End Set
End Property

<DefaultValue(True)> _
Public Property MultiLine() As Boolean
    Get
        Return box.AcceptsReturn
    End Get
    Set(ByVal value As Boolean)
        box.AcceptsReturn = value
    End Set
End Property

<DefaultValue(True)> _
Public Property WordWrap() As Boolean
    Get
        Return box.TextWrapping <> TextWrapping.Wrap
    End Get
    Set(ByVal value As Boolean)
        If value Then
            box.TextWrapping = TextWrapping.Wrap
        Else
            box.TextWrapping = TextWrapping.NoWrap
        End If
    End Set
End Property

<DesignerSerializationVisibility(DesignerSerializationVisibility.Hidden)> _
Public Shadows Property Child() As System.Windows.UIElement
    Get
        Return MyBase.Child
    End Get
    Set(ByVal value As System.Windows.UIElement)
        '' Do nothing to solve a problem with the serializer !!
    End Set
End Property

End Class
```

3. Reconstruire la solution.
4. Ajouter un nouveau formulaire.
5. Recherchez la boîte à outils pour votre nom de classe. Cet exemple est "Vérification orthographique". Il devrait être listé sous les composants "YourSoulutionName".
6. Faites glisser le nouveau contrôle sur votre formulaire
7. Définissez l'une des propriétés mappées dans l'événement de chargement de formulaires

```
Private Sub form1_Load(sender As Object, e As EventArgs) Handles Me.Load
    spellcheckbox.WordWrap = True
    spellcheckbox.MultiLin = True
    'Add any other property modifiers here...
End Sub
```

7. La dernière chose à faire est de modifier la connaissance DPI de votre application. C'est parce que vous utilisez l'application WinForms. Par défaut, toutes les applications WinForms sont DPI UNAWARE. Une fois que vous exécutez un contrôle qui possède un hôte d'élément (WPF Interop), l'application devient désormais DPI AWARE. Cela peut ou peut ne pas gâcher vos éléments d'interface utilisateur. La solution consiste à forcer l'application à devenir DPI UNAWARE. Il y a 2 façons de le faire. Le premier est à travers le fichier manifeste et le second à le coder en dur dans votre programme. Si vous utilisez OneClick pour déployer votre application, vous devez le coder en dur, ne pas utiliser le fichier manifeste ou des erreurs inévitables.

Les deux exemples suivants peuvent être trouvés à l' adresse suivante: Mise à l'échelle WinForms à des paramètres DPI importants - Est-ce possible? Merci à Telerik.com pour l'excellente explication sur DPI.

> Exemple de code DPI codé en dur. Ceci DOIT être exécuté avant que le premier formulaire ne soit initialisé. Je place toujours cela dans le fichier ApplicationEvents.vb. Vous pouvez accéder à ce fichier en cliquant avec le bouton droit sur le nom de votre projet dans l'explorateur de solutions et en choisissant "Ouvrir". Ensuite, choisissez l'onglet de l'application sur la gauche, puis cliquez sur "Afficher les événements de l'application" dans le coin inférieur droit de la liste déroulante de l'écran de démarrage.

```
Namespace My

    ' The following events are available for MyApplication:
    '
    ' Startup: Raised when the application starts, before the startup form is created.
    ' Shutdown: Raised after all application forms are closed.  This event is not raised if
the application terminates abnormally.
    ' UnhandledException: Raised if the application encounters an unhandled exception.
    ' StartupNextInstance: Raised when launching a single-instance application and the
application is already active.
    ' NetworkAvailabilityChanged: Raised when the network connection is connected or
disconnected.
    Partial Friend Class MyApplication

    Private Enum PROCESS_DPI_AWARENESS
        Process_DPI_Unaware = 0
        Process_System_DPI_Aware = 1
        Process_Per_Monitor_DPI_Aware = 2
    End Enum

    Private Declare Function SetProcessDpiAwareness Lib "shcore.dll" (ByVal Value As
PROCESS_DPI_AWARENESS) As Long

    Private Sub SetDPI()
        'Results from SetProcessDPIAwareness
        'Const S_OK = &H0&
        'Const E_INVALIDARG = &H80070057
```

```
        'Const E_ACCESSDENIED = &H80070005

        Dim lngResult As Long

        lngResult = SetProcessDpiAwareness(PROCESS_DPI_AWARENESS.Process_DPI_Unaware)

    End Sub

    Private Sub MyApplication_Startup(sender As Object, e As
ApplicationServices.StartupEventArgs) Handles Me.Startup
        SetDPI()
    End Sub

End Namespace
```

Exemple de manifeste

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0"
xmlns:asmv3="urn:schemas-microsoft-com:asm.v3" >
   <asmv3:application>
        <asmv3:windowsSettings xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">
             <dpiAware>true</dpiAware>
        </asmv3:windowsSettings>
   </asmv3:application>
</assembly>
```

Lire WinForms SpellCheckBox en ligne: https://riptutorial.com/fr/vb-net/topic/8624/winforms-spellcheckbox

Crédits

S. No	Chapitres	Contributeurs
1	Premiers pas avec le langage Visual Basic .NET	Bjørn-Roger Kringsjå, Cary Bondoc, Community, Happypig375, Harjot, Jonathan Nixon, Martin Verjans, MDTech.us_MAN, Misaz, Natalie Orsi, Nico Agusta, Ryan Thomas, StardustGogeta, Virtual Anomaly
2	Accès aux données	MatVAD, Mike Robertson, Nico Agusta
3	au hasard	David Wilson
4	BackgroundWorker	Jones Joseph, Shayan Toqraee
5	Compression de fichiers / dossiers	Cody Gray, MatVAD, Misaz, vbnet3d
6	Conditions	Allen Binuya, Chetan Sanghani, Nathan Tuggy, Robert Columbia
7	Console	Bugs, InteXX, lucamauri, Martin Soles, Matthew Whited, Sam Axe, Slava Auer, StardustGogeta, vbnet3d, VortixDev
8	Conversion de type	Cary Bondoc, LogicalFlaps, vbnet3d
9	Déboguer votre application	Martin Verjans
10	Déclaration de variables	Cody Gray, Darren Davies, Fütemire, glaubergft, keronconk, LogicalFlaps, MatVAD, RamenChef, Sehnsucht
11	Des classes	Darren Davies, Harjot
12	Des listes	Dman, DrDonut, Fütemire, Luke Sheppard, Robert Columbia, Seandk
13	Dictionnaires	DrDonut, Nathan, Proger_Cbsk, Sehnsucht, void
14	En boucle	CiccioRocca, debater, Imran Ali Khan, Mark, MatVAD, Sam Axe, Scott Mitchell, SilverShotBee, TyCobb, vbnet3d, void
15	Enum	4444, CiccioRocca, David Sdot, djv, ElektroStudios, kodkod, LogicalFlaps, Shog9, Steven Doggart, void
16	Filetage	BiscuitBaker, Stefano d'Antonio, Visual Vincent

17	Fonctionnalités de Visual Basic 14.0	Adam Zuckerman, Bjørn-Roger Kringsjå, Blackwood, Crazy Britt, Drake, Fütemire, Gridly, jColeson, liserdarts, Matt Wilko, Nadeem_MK, Nitram, Sam Axe, Stefano d'Antonio, yummypasta
18	GDI +	Dman
19	Génériques	JDC
20	Google Maps dans un formulaire Windows	anonymous, Carlos Borau
21	Introduction à la syntaxe	Bugs, Mark Hurd, Martin Verjans, mnoronha, Nat G., Nico Agusta, Sehnsucht
22	La gestion des erreurs	Adam Zuckerman, Bjørn-Roger Kringsjå, Happypig375, Luke Sheppard, MatVAD, Nico Agusta, Vishal
23	La gestion des fichiers	Dan Granger, Luke Sheppard, Matt Wilko, Misaz, Shayan Toqraee, vbnet3d
24	Lecture du fichier texte compressé à la volée	Proger_Cbsk
25	Les fonctions	Berken Usar
26	Les opérateurs	Bjørn-Roger Kringsjå, Cary Bondoc, MatVAD, Mike Robertson, Pasilda, Robert Columbia, RoyalPotato, Sam Axe , Sree, varocarbas, void
27	Liaison de données WPF XAML	Milliron X
28	LINQ	Dan Drews, Daz, Derek Tomes, Fütemire, H. Pauwelyn, Mark Hurd, Misaz, Sam Axe, Sehnsucht, Zev Spitz
29	Manipulation de connexion	Jonas_Hess
30	Méthodes d'extension	Fütemire, InteXX, Matt Wilko, Sam Axe, Stefano d'Antonio, void
31	Modèle asynchrone basé sur des tâches	Stefano d'Antonio
32	Mots-clés ByVal et ByRef	Adam Zuckerman, Misaz, Sehnsucht
33	Mots-clés OOP	4444, David, JDC, Matt Wilko, Nat G.

34	Multithreading	4444, Daz, MatVAD
35	NullReferenceException	Alessandro Mascolo, RamenChef, Sehnsucht
36	Objets jetables	KE0GSD, Mark Hurd, Martin Verjans, Matt Wilko, Misaz, Mithrandir, Sam Axe
37	Opérateurs de court-circuit (et aussi - ou moins)	Bart Jolling, CiccioRocca, Kendra, Sam Axe
38	Option explicite	Happypig375, Matt Wilko, sansknwoledge
39	Option Infer	Alex B., LogicalFlaps, Mark Hurd
40	Option Strict	Andrew Morton, Cary Bondoc, Matt Wilko, RussAwesome, Sam Axe, vbnet3d
41	Récursivité	Robert Columbia
42	Réflexion	Axarydax, Matt, Sam Axe, void
43	Rendez-vous amoureux	Matt Wilko, Misaz
44	Serveur ftp	Misaz
45	Tableau	BunkerMentality, djv, Drarig29, Luke Sheppard, Mark Hurd, MatVAD, Robert Columbia, Ryan Thomas, Sam Axe, Sehnsucht, Steven Doggart, TuxCopter, vbnet3d, VortixDev, zyabin101
46	Test d'unité dans VB.NET	wbadry
47	Travailler avec Windows Forms	djv, SilverShotBee, vbnet3d
48	Utiliser axWindowsMediaPlayer dans VB.Net	Berken Usar
49	Utiliser BackgroundWorker	MatVAD
50	Utiliser la déclaration	VV5198722
51	WinForms SpellCheckBox	Nathan