

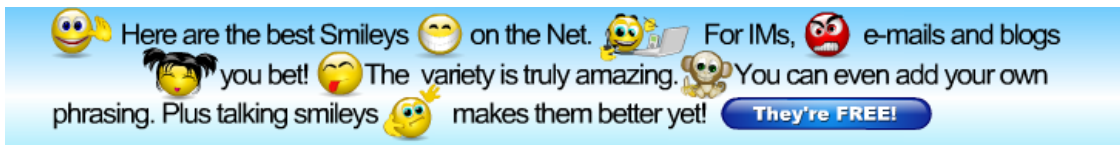
Microsoft Visual Basic

2008

Fundamentals	Application Design	Applications Accessories
Introduction to MSVB	Introduction	Introduction to the Main Menu
Introduction to Methods	Visual Control Addition	Contextual Menus
Dynamic Control Creation	Selecting a Control	Characteristics of Menu Items
	Moving a Control	
	Control Alignment	
	Resizing Controls	

Studio Windows	Functions and Procedures	Exception Handling
The Toolbox	Message Box	Error Handling
The Properties Window	Input Box	Introduction to Exceptions
		.NET Support for Exceptions
		Techniques of Using Exceptions

File Processing		
Fundamentals	Details	Serialization
Introduction	File Information	Binary
Writing to a Stream	File System Information	SOAP
Reading From a Stream	Directories	
Exception Handling		

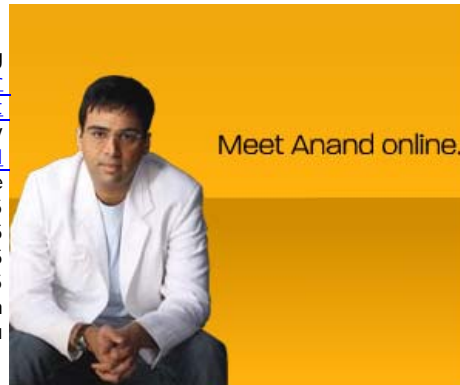


Introduction to Microsoft Visual Basic

Microsoft Visual Basic Fundamentals

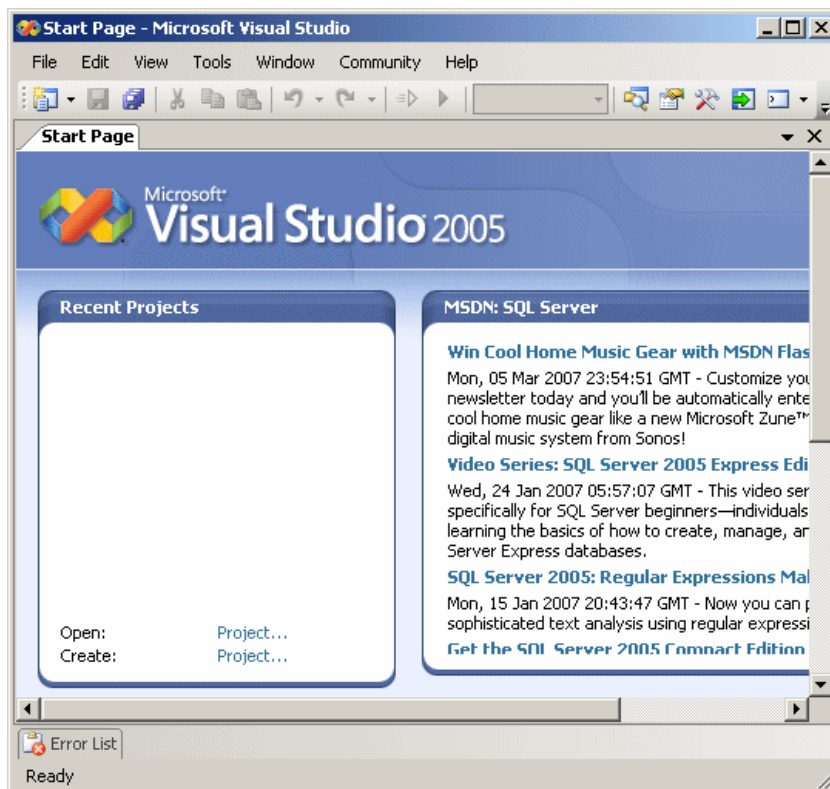
Startup

Microsoft Visual Basic is a programming environment used to create [graphical user interface](#) (GUI) [applications](#) for the [Microsoft Windows](#) family of [operating systems](#). It usually ships either by itself or as part of [Microsoft Visual Studio](#). To follow these lessons, you must have installed either Microsoft Visual Basic 2005 Express Edition, Microsoft Visual Basic 2005 Professional, or Microsoft Visual Studio 2005 (Professional). To get Microsoft Visual Basic 2005 Express Edition, you can download it free from the [Microsoft](#) web site. After downloading it, you can install it.



From now on, unless specified otherwise, we will use the expressions "Microsoft Visual Basic" or "Visual Basic" to refer to Microsoft Visual Basic 2005.

After installing it, to use Microsoft Visual Basic, you must launch. To launch Microsoft Visual Basic 2005 Express Edition, you can click Start -> (All) Programs -> Microsoft Visual Basic 2005 Expression Edition. If you are using Microsoft Visual Studio 2005 Professional, to start it, on the task bar, you can click Start -> (All) Programs -> Microsoft Visual Studio 2005 -> Microsoft Visual Studio 2005:



The Microsoft Visual Basic Interface

Microsoft Visual Basic presents itself as a series of tools used to assist you in creating computer

programs. As a normal Windows application, it starts on top with a menu and some toolbars. It is also equipped with various windows, considered as tools, you will be using. Most of these tools are available or are functional only if you have primarily created or opened a project.

Ads by Google

[Distance Learning Courses](#)

UK University
Qualifications Browse and
Apply Online Today
www.rdi.co.uk/Distance_Learning

[Visual Basic 6.0 Codes](#)

Find Solutions for your
Business. Free Reports,
Info & Registration!
www.KnowledgeStorm.com

[Free Movie Preview API](#)

100000 videos with
metadata to make widgets,
gadgets and applications
InternetVideoArchive.com

[Free SQL Server Tips](#)

MS SQL Server Tips &
Techniques Solve SQL
problems fast and free
www.mssqltips.com

[Introduction to Fieldbus](#)

Free online courses make
learning fieldbus easy and
convenient - 24/7
www.PlantWebUniversity.com

❖ Practical Learning: Starting Microsoft Visual Basic

- To launch Microsoft Visual Basic:
If are using Microsoft Visual Basic 2005 Express Edition, on the taskbar, click Start -> (All) Programs -> Microsoft Visual Basic 2005 Express Edition
If you are using Microsoft Visual Studio 2005, on the task bar, click Start -> (All) Programs -> Microsoft Visual Studio 2005 -> Microsoft Visual Studio 2005

www.windev.com

Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)

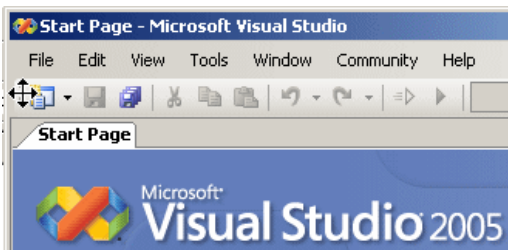


Introduction to Microsoft Visual Basic

The Studio Windows

The Toolbars

A toolbar is an object made of buttons. These buttons provide the same features you would get from the (main) menu, only faster. Under the main menu, the [IDE](#) is equipped with the Standard toolbar. By default, the Standard toolbar is positioned under the main menu but you can position it anywhere else on the IDE. To move a toolbar, position the mouse on the dotted line on its left section. The mouse pointer will change into a cross:



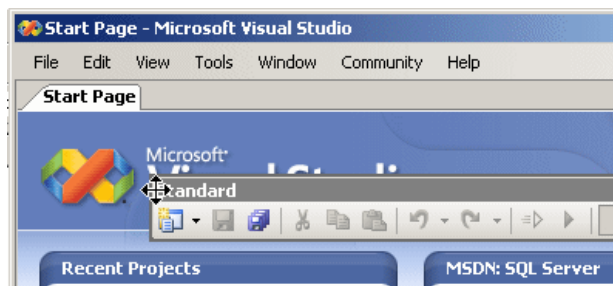
- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares
- 10 Sub-Domains
- All Databases
- POP3/Webmail
- Full Control Panel

Rs.2000/- per year

MANASHOSTING

www.manashosting.com Ads by Google

Then click and drag away from that position:



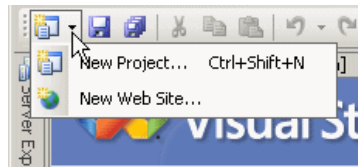
In the same way, you can position the toolbar anywhere on the screen. You can also attach or "dock" it to one of the four sides of the IDE. When a toolbar is not docked to one side of the IDE, it is said to float. When a toolbar is floating, you can resize it by dragging one of its borders. If a toolbar is floating, to put it back to its previous position, you can double-click its title bar.

By default, when you start Microsoft [Visual Studio](#), it is equipped with one toolbar: Standard. To get more toolbars, on the main menu, you can click View -> Toolbars and click the toolbar of your choice. You can also right-click any available toolbar or the main menu. This displays a list of all the available toolbars. Those that are currently opened have a check mark next to them.

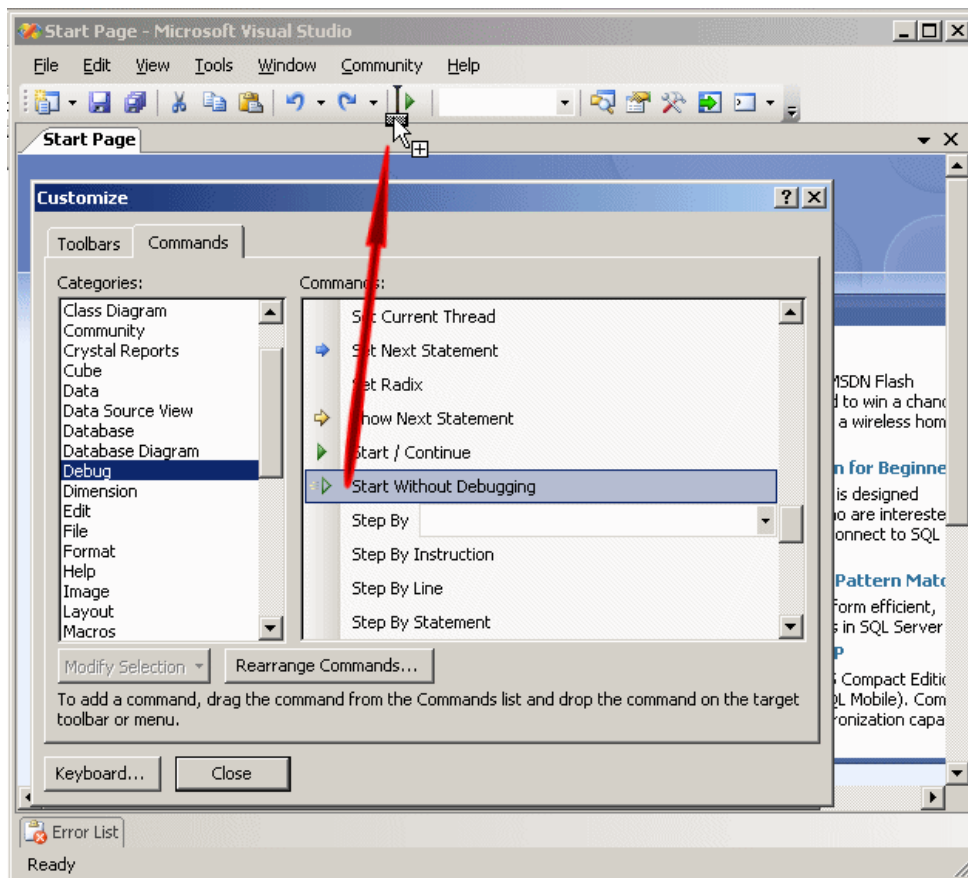
A toolbar is equipped with buttons that could be unfamiliar. Just looking at one is not obvious. To know what a button is used for, you can position the mouse on top of it. A [tool tip](#) will come up and display for a few seconds.

In our lessons, each button on any toolbar will be named after its tool tip. This means that, if a tool tip displays "Hungry", its button will be called the Hungry button. If a tool tip displays "Save All", its button will be called the Save All button. If you are asked to click a button, position your mouse on different buttons until one displays the referred to name.

Some buttons present an arrow on their right side. This arrow represents a menu. Here is an example:



Like the menu, the toolbars can be customized. To customize the Standard toolbar by adding buttons to it, you can right-click anything on the main menu or the toolbar and click Customize... On the Customize dialog box, you can click the Commands tab. In the Categories list, you can click a category, such as Debug. In the Commands list, you can click and drag an item, position it somewhere in the Standard toolbar, and release the mouse. Here is an example:



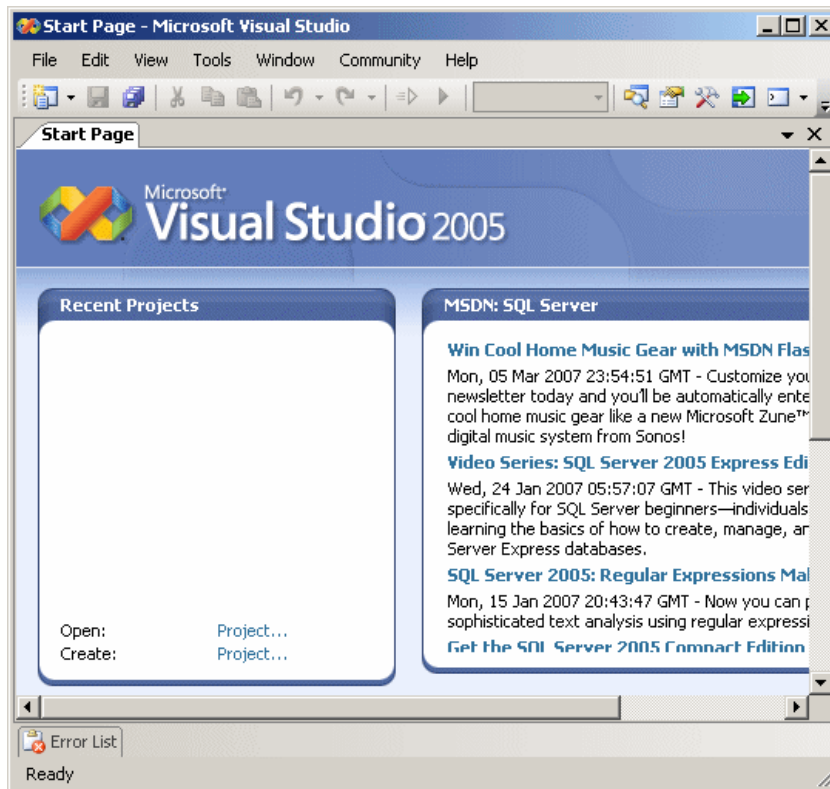
When you have finished, you can click the Close button on the Customize dialog box

The Start Page

The [Start Page](#) is the first wide area that appears when [Microsoft Visual Studio](#) comes up. The section displays a title as Recent Projects. At any time, to display the Start Page:

- You can click its tab on the left side just under the Standard toolbar
- On the main menu, you can click View -> Other Windows -> Start Page
- On the main menu, you can click Windows -> Start Page

If you have just installed Microsoft Visual Studio or have not previously opened a project, the Recent Projects section would be empty. Here is an example:




Once you start creating and using projects, they display in the Recent Projects section by their names.

The middle section allows you to check new articles from [Microsoft](#) and partners directly from Visual Studio 2005 through an [Internet connection](#).

Showing and Closing a Window

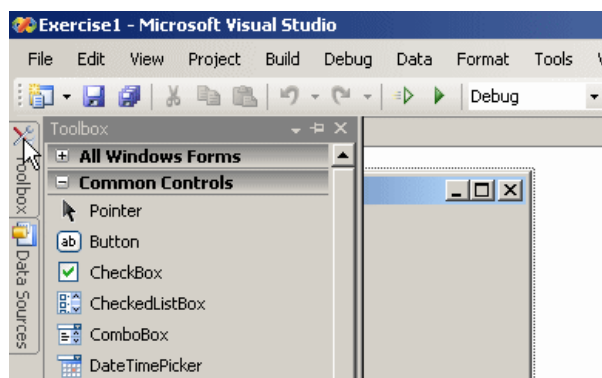
When you start or open a project, Microsoft Visual [Studio 2005](#) makes some windows available. These are the most regularly used windows. If you think that one of them is not regularly used in your types of assignments, you can remove it from the screen. To hide a window:

- You can click its Close button 
- You can click its title bar and click Hide

All of the windows you can use are listed in the View menu. Therefore, if a window is not displaying, you can click View on the main menu and click a window of your choice.

Auto Hiding a Window

When creating your [applications](#), you will use a set of windows that each accomplishes a specific purpose. Some windows are represented with an icon but hide the rest of the body. To display such a window, you can position the mouse on it. This would expand the window:



If you expand a window, it would display a title bar with two buttons. One is called Auto Hide and the other is the classic Close button:

Ads by Google

[WPF Dockable Windows](#)

Add rich window management including MDI and 3D to your apps
www.divelements.co.uk

[Free Windows XP IE Skins](#)

Excite Your Internet Explorer with Crawler Toolbar Skins. Free!
www.CrawlerTools.com

[Shell Control Pack](#)

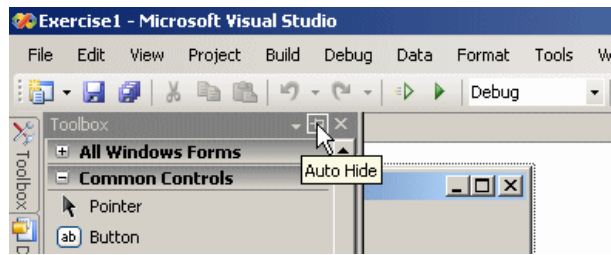
Windows Explorer controls for Delphi, VCL, ActiveX and .NET!
www.plasmatech.com


[New Address Bar for IE](#)

Stop wasting browser space for search bars. Free IE toolbar.
www.quero.at

[Toolbar](#)

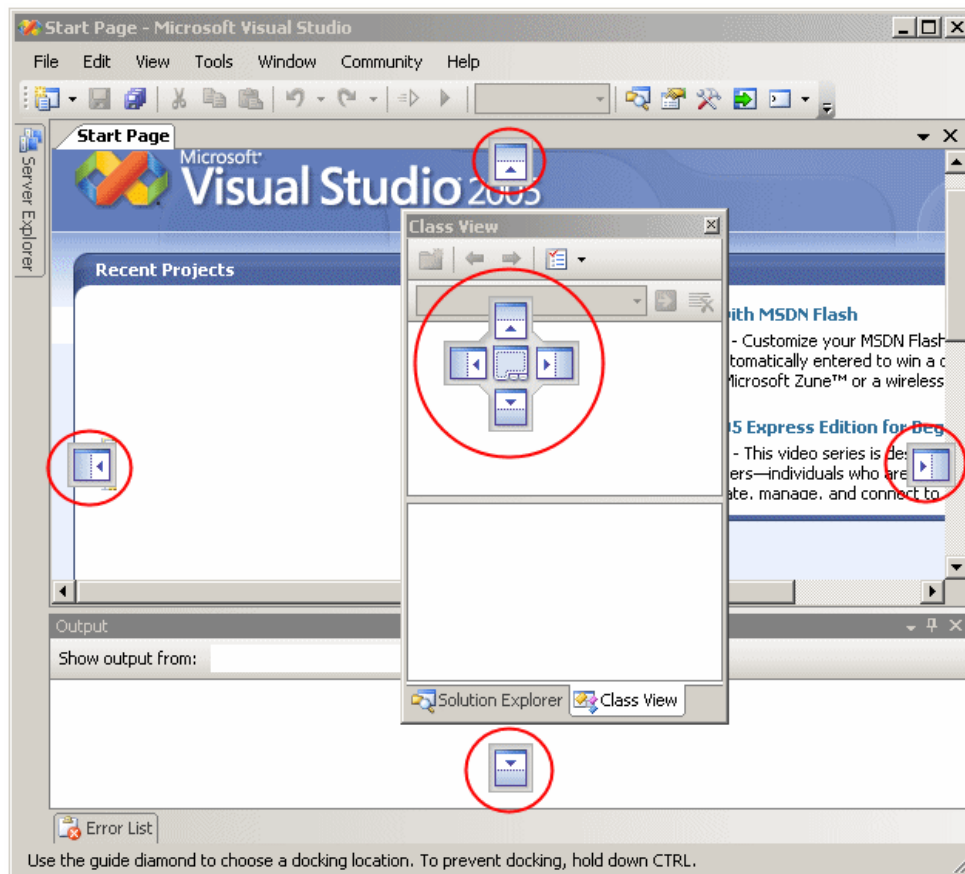
Let The Experts Help. Live Support 100% Free Of Charge.
www.Fixya.com/Toolbar



If you expand a window but find out you don't need it at that position, you can just move the mouse away from it. The window would return to its previous state. Based on this functionality, if you are [working](#) with a window and move the mouse away from it, it would retract. If you need it again, you would have to reopen it using the same technique. If you are going to work with a certain window for a while, you can keep it open even if you move the mouse away. To do this, you can click the Auto Hide button. If clicked, the Auto Hide button changes from pointing left to pointing down .

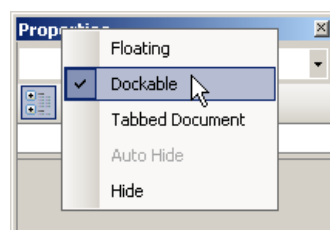
Dockable Windows

By default, [Visual Studio 2005](#) installs some windows to the left and some others to the right of the screen. You can change this arrangement if you want. To do this, expand a window, then click its title bar and start dragging. When you do this, the screen would display 5 buttons: one to each side and one in the middle:



To position a window on one side of the screen, drag its title bar to one of the four buttons on the sides.

You can dock a window only if it is dockable. To make sure that a window is dockable, you can right-click its title bar and click Dockable:



If you don't want the window to be dockable, you can right-click its title bar and click Floating.

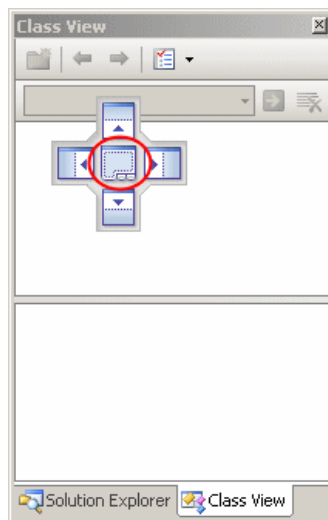
Floating Windows

Most of the windows you will use are positioned on one side of the screen. If you want, you can have a window that stays on top of other window but cannot be "glued" to one side. Such a window is said to float. To float a window, drag its title bar and release it somewhere in the middle of the screen but not on one of the previously mentioned buttons because, while dragging, if you release the mouse on one of the buttons, and if the window is dockable, it would assume the position of where you released the mouse.

If you don't want a window to be dockable and you only want it to float, right-click its title bar and click Floating.

Tabbing a Window

Instead of accessing a window from one side of the screen or from its sharing an area with another window, you can make it display a tab. To do this, drag its title bar and release the mouse when its gets to the middle button that displays some tabs:



When a window is tabbed, you cannot drag its tab to position it on one side of the screen. If you want to remove it from its tabbing position, first right-click its tab and click either Floating or Dockable.

Coupling Windows

You can make two or more windows share one side of the screen or to share an area. To do this, drag its title bar to the window whose area you want to share, then position the mouse on the middle button and release it.

Unlimited Webspace and Unlimited Bandwidth		FREE	Only Rs.2000/-
⇒ 100 Email Ids of 10 GB	⇒ Full Control Panel	Domain Registration	
⇒ All Database FREE	⇒ 200 SEO Tools/Softwares/Weblinks		
⇒ 5000 Templates / Website Builder			

www.manashosting.com Feedback - Arts hv Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Introduction to Microsoft Visual Basic

A Project

Introduction

To create a computer program, also called an application, you create a series of files and group them in an ensemble called a project. This contains various modules, files, assemblies (or libraries), and resource files.

Creating a Project

A typical application consists of more than one module and can even be as complex as you want. To make it faster and a little easier to graphically create an application, you would need a good functioning environment like Microsoft [Visual Basic](#). Using it, you can create a new project or you can open an existing one.

● One Domain FREE
 ● Unlimited Webspace
 ● Unlimited Bandwidth
 ● 100 E-mail Ids
 ● 5000 Templates
 ● Website Builder
 ● 500 Source Code
 ● 200 SEO Tools
 ● 100 Softwares
 ● 10 Sub-Domains
 ● All Databases
 ● POP3/Webmail
 ● Full Control Panel

Rs.2000/-
per year

MANASHOSTING

www.manashosting.com Ads by Google

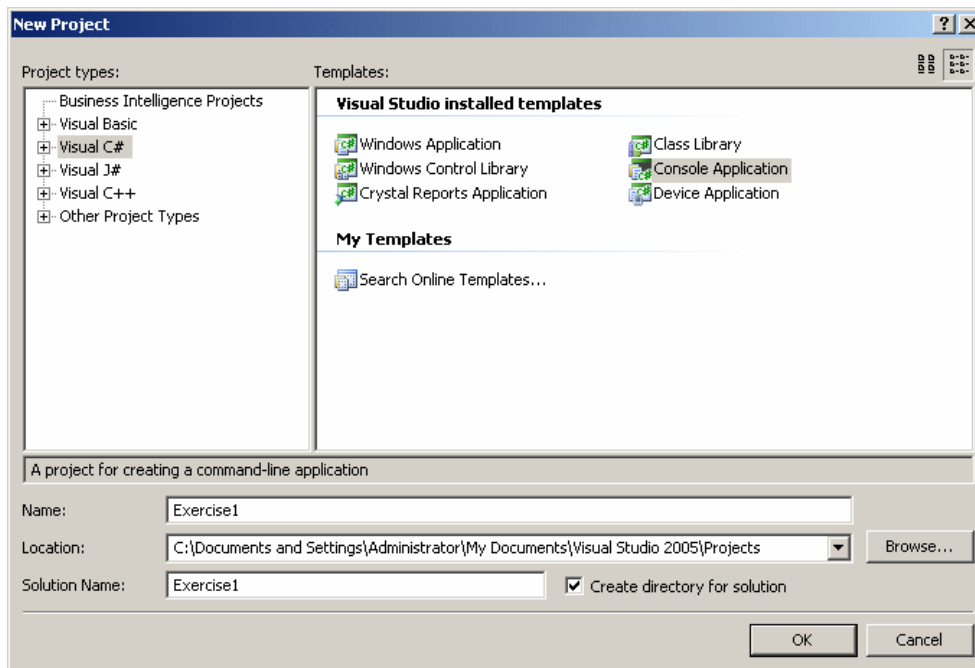
To create a Visual Basic project, you can display the New Project dialog box. To open the New Project dialog box:

- On the Start Page, on the right side of Project, click Create...
- If you are using Microsoft Visual Basic 2005 Express Edition, on the main menu, you can click File -> New Project... If you are using Microsoft Visual Basic 2005 Professional, on the main menu, you can click File -> New -> Project...
- On the Standard toolbar, you can click the New Project button 
- You can press Ctrl + Shift + N

In the New Project dialog box, select Visual Basic Projects, select the type of project, give it a name, specify its directory, and click OK.

❖ Practical Learning: Creating a Project

1. On the main menu, click File -> New Project or File -> New -> Project...
2. In the Templates section, click Console Application
3. In the Name edit box, type **Exercise1**




4. Accept the name in the Location text box and click OK. This creates a new project

Compiling and Executing a Project

The instructions created for a Visual Basic project are written in plain English in a language easily recognizable to the human eye. After creating the file(s) of a project, you would compile the project to get an executable that becomes ready to be distributed to your users.

To compile and execute a project in one step, on the main menu, you can click Debug -> Start Without Debugging. Although there are other techniques or details in compiling (or debugging) and executing a project, for now, this is the only technique we will use until further notice.

❖ Practical Learning: Executing a Project

1. To execute the application, on the main menu, click Build -> Build Exercise1
2. To execute the application, on the Standard toolbar, click the Start Debugging button 

Opening a Project

As opposed to creating a new project, you can open a project that either you or someone else created. To open an existing project:

- On the Start Page, on the right side of Project, click Open...
- If you are using Microsoft Visual Basic 2005 Express Edition, on the main menu, you can click File -> Open Project... If you are using Microsoft Visual Basic 2005 Professional, on the main menu, you can click File -> Open -> Project...
- You can press Ctrl + Shift + O

This action would display the Open Project dialog box. This allows you to select a project and open it.

Overview of GUI Applications

Introduction

Microsoft Visual Basic is a programming environment that allows you to create various types of applications. In our lessons, we will mostly create graphical applications, also called Windows applications or Windows Forms applications.

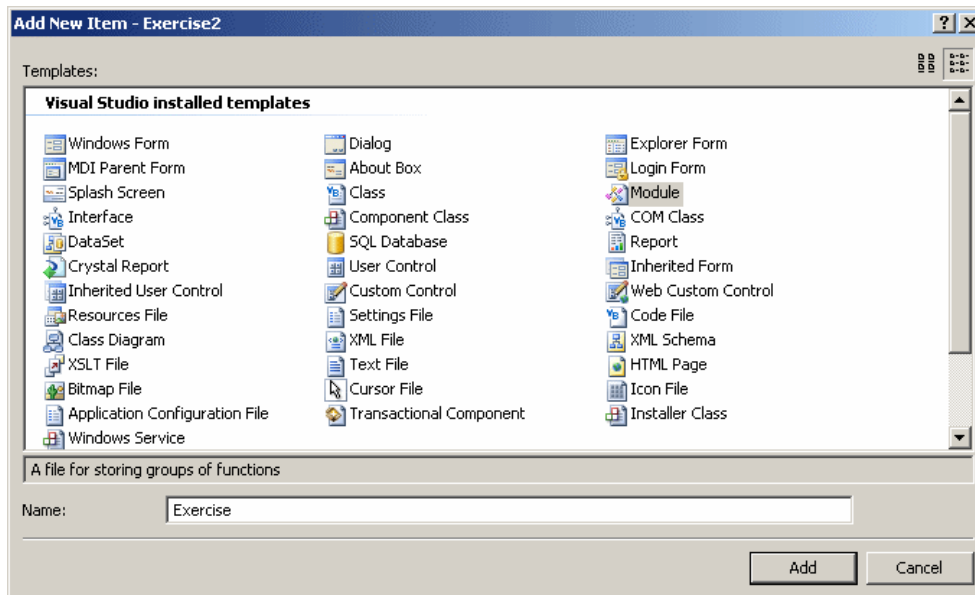
A Windows application primarily appears as a rectangular object that occupies a portion of the screen. This type of object is under the management of the operating system, [Microsoft Windows](#). Based on the functionality of Microsoft Windows, for an application to become useful, it must be opened. An application must have an entry point. On a C/C++ application, this entry point is a function called **main**. On a Win32 application, this entry point is a function called **WinMain**. In the Visual Basic language, this entry point is a function named **Main**.

❖ Practical Learning: Starting a GUI Application

1. To create a new application, on the main menu, click File -> New Project or File -> New ->

Project...

2. In the Project Types list, expand Visual Basic and click Windows. In the Templates section, click Empty Project
3. In the Name edit box, type **Exercise2** and click OK
4. On the main menu, click Project -> Add New Item...
5. In the Templates list, click Module
6. Change the Name to **Exercise**



7. Click Add
8. Change the contents of the file as follows:

```
Module Exercise
```

```
Function Main() As Integer
    Return 0
End Function
```

```
End Module
```

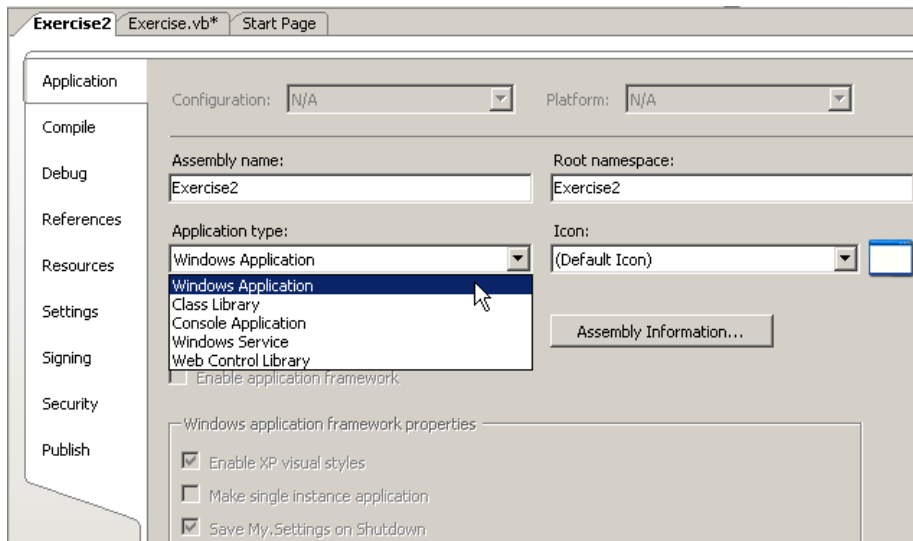
Windows Application Configuration

Although you can directly create a graphical application when starting your project, if you had created a console application, you can still easily transform it into a Forms application:

- The most required action consists of changing some characteristics of the project. To take care of this, on the main menu, you can click Project -> *ProjectName* Properties... and then change the value of the Output Type combo box to Windows Application
- Before or after setting the Output Type to Windows Application, you can create a form

❖ Practical Learning: Configuring a Windows Application

1. On the main menu, click Project -> Exercise2 Properties...
2. On the left side, make sure Application is selected. In the right section, click the arrow of the **Output Type** combo box and select **Windows Application**



3. Save and close the window

Forms Fundamentals

Windows Forms is a technique of creating computer applications based on the common language runtime (CLR). It offers a series of objects called Windows Controls or simply, controls. These controls are already created in the .NET Framework through various [classes](#). Application programming consists of taking advantage of these controls and customizing them for a particular application. To exploit these controls and other features of the .NET Framework, there are various types of applications you can create, including graphical applications (Windows Application), web-based applications ([ASP.NET Web Application](#)), console applications (Console Application), etc.

The objects used in a Windows application are stored in libraries also called assemblies. As normal libraries, these assemblies have the extension .dll (which stands for dynamic link library). In order to use one of these objects, you must know the name of the assembly in which it is stored. Then you must add a reference to that assembly in your application.

To add a reference to an assembly, on the main menu, you can click Project -> Add Reference... You can also right-click the name of the project in the Solution Explorer and click Add Reference... Any of these actions would display the Add Reference dialog box from where you can click the reference, click Select and click OK. If you don't see the reference you are looking for, you can locate it on another drive or directory using the Browse button.

There are two broad categories of objects used in a Windows Forms application: the forms and the controls. A form is the most fundamental object used on an application. It is a rectangular object that uses part of the computer [desktop](#) to represent an application. A form is based on the **Form** class that is defined in the **System.Windows.Forms** namespace created in the **System.Windows.Forms.dll** assembly. Every GUI application you will create starts with a form. There are various techniques you can use to get a form in your application:

- You can programmatically and manually create a form
- You can inherit a form from the **Form** class
- You can create a form based on another form that either you or someone else created already, etc.

The primary means of getting a form into an application consists of deriving one from the **Form** class.

❖ Practical Learning: Deriving a Form From the Form Class

1. To add a reference to the assembly in which the **Form** class is defined, on the main menu, click Project -> Add Reference...
2. In the Add Reference dialog box, click the .NET tab if necessary and scroll down in the list
3. Click System
4. Press and hold Ctrl
5. Click System.Windows.Forms

MANASHOSTING



MANASHOSTING

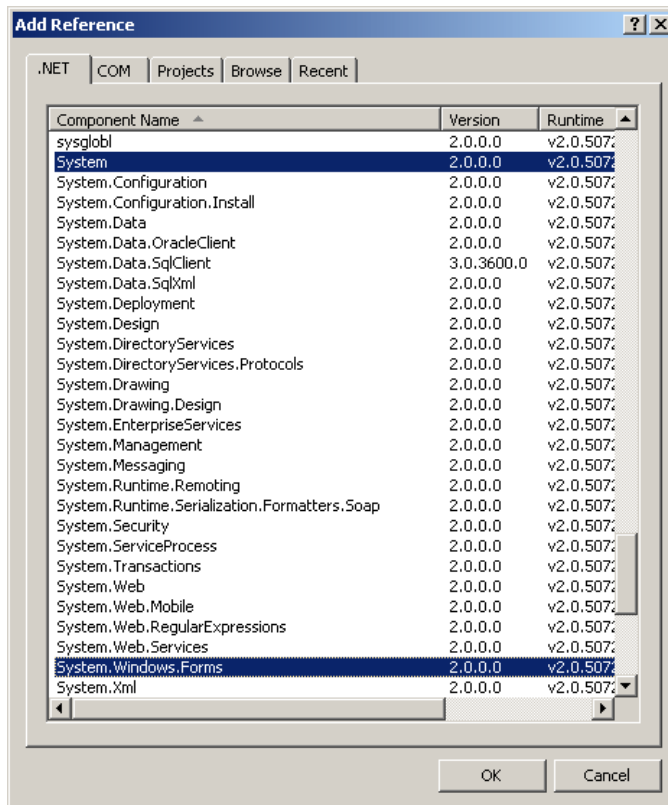
Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >

www.manashosting.com
Ads by Google



- Click OK
- To inherit a form from the **Form** class, change the file as follows:

```
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

    End Class

    Function Main() As Integer
        Return 0
    End Function

End Module
```

- Save the file

The Application Class

The form is the object that gives presence to an application. Once you have created the (primary) form of your application, you can get it ready to display on the screen. This is taken care of by the **Application** class equipped to start an application, process its messages or other related issues, and stop the application.

The **Application** class provides the overloaded **Run()** method that can be used to start a program. One of the versions of this method takes a form as argument. This form must be the first, main or primary form of your application; it will be the first to display when the application comes up.

❖ Practical Learning: Using the Application Class

- To prepare the [application](#) for starting, change the **Main()** method as follows:

```
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

    End Class

    Function Main() As Integer

        'Instantiate an Program object
        Dim frmStart As Starter

    End Function

End Module
```

```

' Allocate memory for the object, using the new operator
frmStart = New Starter

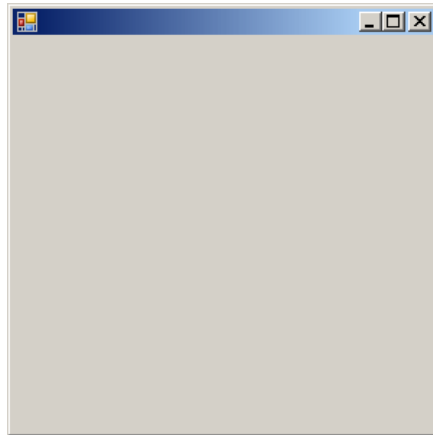
' Call the Run() static method of the Application
' and pass it the instance of the class to display
Application.Run(frmStart)


Return 0
End Function

End Module

```

2. Test the application



3. Close it by clicking its system Close button  and return to your programming environment

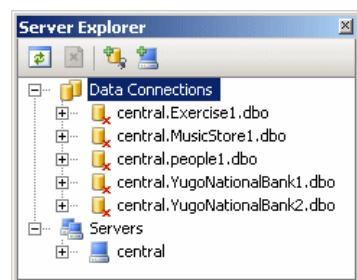
The Project Interface

Introduction

Besides the windows and functionalities we reviewed earlier, when you work on a project, there are other features that become available.

The Server Explorer

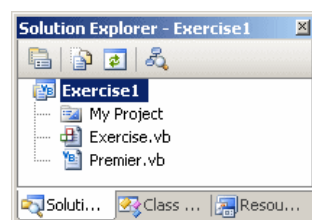
The Server Explorer is an accessory that allows you to access [SQL Server](#) databases without using the physical [server](#) and without opening Microsoft SQL Server:



The items of this window display in a tree. To expand a node, you can click its + button. To collapse it, click its - button.

The Solution Explorer

The Solution Explorer is a window that displays the file names and other items used in your project:



The items of this window display in a tree. To expand a node, you can click its + button. To collapse it, click its - button. To explore an item, you can double-click it. The result depends on

the item you double-clicked.

The Solution Explorer can be used to create a new class, a new folder, or a reference. To perform any of these operations, you can right-click a folder node such as the name of the project, position the mouse on Add and select the desired operation. You can also perform any of these operations from the Project category of the main menu.

Besides adding new items to the project, you can also use the Solution Explorer to build the project or change its properties. If you add one or more other project(s) to the current one, one of the projects must be set as the default. That project would be the first to come up when the user opens the application. By default, the first project created is set as the default. If you have more than one project, to set the default, right-click the name of the desired project in Solution Explorer and click Set As StartUp Project.

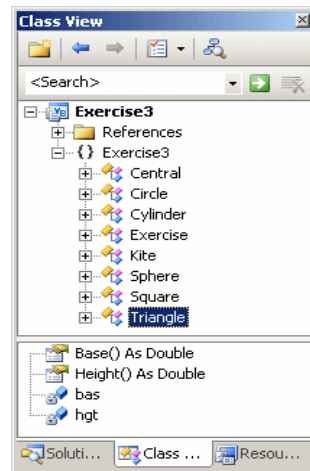
The Solution Explorer also you to rename or delete some of the items that belong to your project.

❖ Practical Learning: Using the Solution Explorer

1. To start a new project, on the main menu, click File -> New -> Project...
2. In the Templates list, click Empty Project and change the Name to **Exercise3**
3. Click OK
4. On the main menu, click Project -> Exercise3 Properties...
5. In the left frame, make sure Application is selected.
In the right frame, click the arrow of the Output Type combo box and select Windows Application
6. If the Solution Explorer is not visible, on the main menu, click View -> Solution Explorer.
In the Solution Explorer, right-click Exercise3 and click Add Windows Form...
7. In the Templates list, make sure Windows Form is selected.
Set the Name to **Exercise** and click Add

The Class View

The Class View displays the various classes used by your project, including their ancestry. The items of the Class View are organized as a tree list with the name of the project on top:



The Class View shares some of its functionality with the Solution Explorer. This means that you can use it to build a project or to add new class.

While the Solution Explorer displays the items that are currently being used by your project, the Class View allows you to explore the classes used in your applications, including their dependencies. For example, sometimes you will be using a control of the of the .NET Framework and you may wonder from what class that control is derived. The Class View, rather than the Solution Explorer, can quickly provide this information. To find it out, expand the class by clicking its + button.

❖ Practical Learning: Using the Class View

1. If the Class View is not visible, on the main menu, click View -> Class View.
In the Class View, expand the Exercise3 node if necessary.
Right-click the name of the project Exercise3 -> Add -> Class...
2. In the Templates list, make sure Class is selected. Change the Name to **Central** and click Add

Unlimited Webspace and Unlimited Bandwidth

FREE
Domain Registration

Only
Rs. 2000/-

- ⇒ 100 Email Ids of 10 GB
- ⇒ All Database FREE
- ⇒ 5000 Templates / Website Builder
- ⇒ Full Control Panel
- ⇒ 200 SEO Tools/Softwares/Weblinks

www.manashostinn.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Introduction to Microsoft Visual Basic

The Code Editor

Introduction

There are two main ways you will manipulate an object of your application, visually or using code. In future sections, we will explore details of visually designing a control. Code of an application is ASCII text-based, written in plain English and readable to human eyes. For an application, you can use any text editor to write your code but one of Visual Studio's main strengths is the code editor. It is very intuitive.

The Code Editor is a window specially designed for code writing.

[Visual Basic Code Library](#)

Open Source Code Snippet Library. Free Community for Developers.

www.daniweb.com/code

[Free UML 2 Design Tool](#)

Visually develop applications with 13 UML 2 Diagrams, ERD, BPM & More!

www.visual-paradigm.com

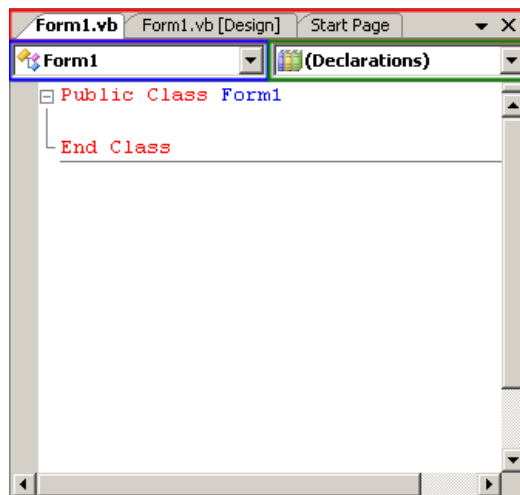


Ads by Google



Although all languages of the Visual Studio programming environment share the Code Editor, once you have started a type of application, the Code Editor is adapted to the language you are using. Its parser (a program used internally to analyze your code) behaves according to the language of your choice. The features and behaviors of the Code Editor are also different, depending on your language.

The Code Editor is divided in 4 sections:



❖ Practical Learning: Introducing the Code Editor

1. Change the file as follows:

```
Imports System.Windows.Forms

Public Class Central

    Public Shared Function main() As Integer

        Application.Run(New Exercise)
        Return 0

    End Function

End Class
```

2. Execute the application to see the new form

3. To create a new class, on the main menu, click Project -> Add Class...
4. Set the Name to **Circle** and click Add

The Tabs Bar

The top section of the Code Editor displays tabs of property pages. Each tab represents a file. To add a new file to the project, on the main menu, you can click

- File -> New -> File...
- Project -> Add New Item...

Once in the Add New Item dialog box, in the Templates section, click the type of file you want to create, type a name in the Name text box, and press Enter. After the file has been created, it is represented by a tab in the top section of the Code Editor. In the same way, you can add as many files as you judge them necessary. To access a tab:

- You can click its name in the Tabs Bar
- On the main menu, you can click Window and click the name of the desired tab

By default, the tabs display in the order their files were created or added to the project, from left to right. If you don't like that arrangement, click and drag its tab either left or right beyond the next tab

❖ Practical Learning: Introducing the Code Editor

1. To create a new class, on the main menu, click Project -> Add Class...
2. Set the Name to **Square** and click Add
3. To create a new class, on the main menu, click Project -> Add Class...
4. Set the Name to **Triangle** and click Add
5. To access the Circle tab, on the main menu, click Window -> Circle.vb
6. Change the file as follows:

```
Public Class Circle
    Private rad As Double

    Public Property Radius() As Double
        Get
            Return rad
        End Get
        Set(ByVal value As Double)
            rad = value
        End Set
    End Property

    Public ReadOnly Property Area() As Double
        Get
            Return rad * rad * Math.PI
        End Get
    End Property
End Class

Public Class Sphere
    Inherits Circle
    Public Overloads ReadOnly Property Area() As Double
        Get
            Return 4 * Radius * Radius * Math.PI
        End Get
    End Property
End Class

Public Class Cylinder
    Inherits Circle

    Private hgt As Double

    Public Property Height() As Double
        Get
            Return hgt
        End Get
        Set(ByVal value As Double)
            hgt = value
        End Set
    End Property
End Class
```

7. To access another tab, in tabs section, click Triangle.vb
8. Change the file as follows:

```
Public Class Triangle
    Private bas As Double
    Private hgt As Double

    Public Property Base() As Double
        Get
            Return bas
        End Get
        Set(ByVal value As Double)
            bas = value
        End Set
    End Property

    Public Property Height() As Double
        Get
            Return hgt
        End Get
        Set(ByVal value As Double)
            hgt = value
        End Set
    End Property
End Class

Public Class Kite

End Class
```

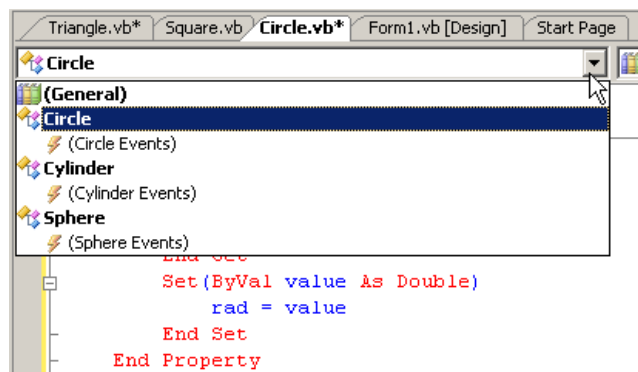
9. Save all

Ads by Google

[ASP.NET OLAP Control](#)

The Class Name Combo Box

The top-left section of the Code Editor displays a combo box named Class Name. As its name indicates, this combo box holds a list of the classes (and structures) that are created in the current file. You can display the list if you click the arrow of the combo box:



Each item of the Class Name combo box displays the name of its type associated with its parent as implemented in the code.

❖ Practical Learning: Using the Types Combo Box

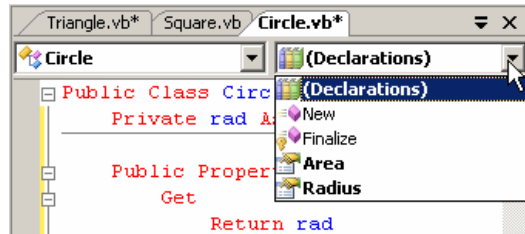
1. Click the Circle tab
2. In the Class Name combo box, select Circle

The Method Name Combo Box

The top-right section of the Code Editor displays a combo box named Members. The Members combo box holds a list of the members of classes. The content of the Members combo box depends on the item that is currently selected in the Class Name combo box. This means that, before accessing the members of a particular class, you must first select that class in the Class Name combo box. Then, when you click the arrow of the Method Name combo box, the members of only that class display:

Want give a Web
app OLAP
functions? Easy with
RadarCube! DBMS
or MSAS.

www.radar-soft.com



If you select an item from the Method Name combo box, the Code Editor jumps to that members and positions the cursor to the left of the member.

❖ Practical Learning: Using the Method Name Combo Box

1. In the Method Name combo box, select Area
2. Press the up arrow key and add a Diameter property as follows:

```
Public Class Circle
    Private rad As Double

    Public Property Radius() As Double
        Get
            Return rad
        End Get
        Set(ByVal value As Double)
            rad = value
        End Set
    End Property

    Public ReadOnly Property Diameter()
        Get
            Return rad * 2
        End Get
    End Property

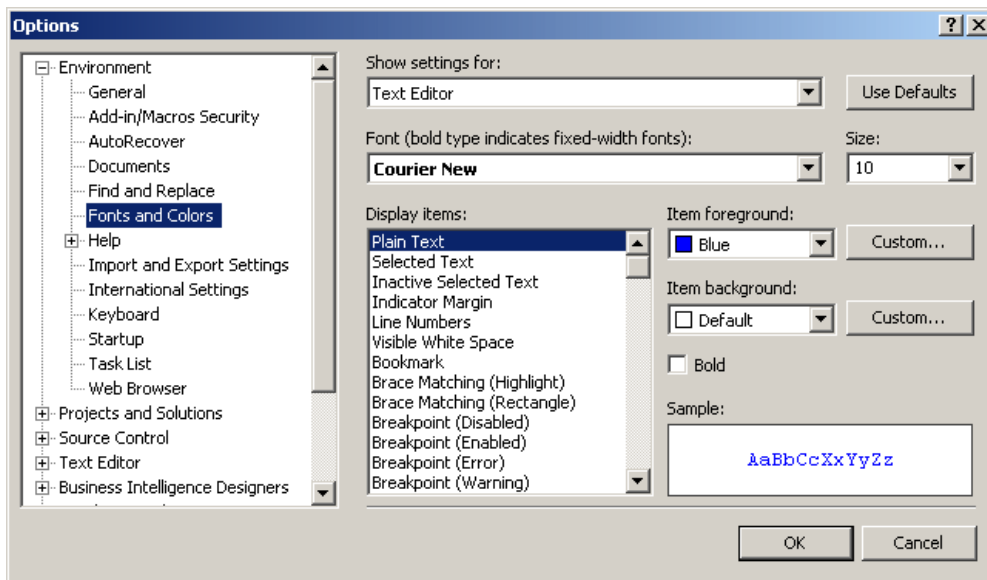
    Public ReadOnly Property Area() As Double
        Get
            Return rad * rad * Math.PI
        End Get
    End Property
End Class

. . . No Change
```

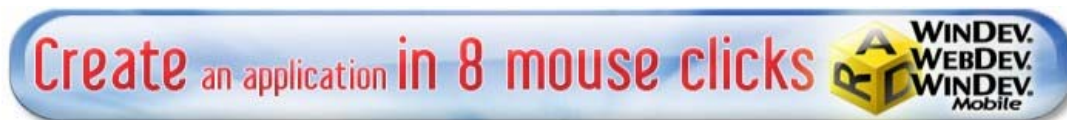
3. Save all

Code Colors

Code is written in a wide area with a white background. This is the area you use the keyboard to insert code with common readable characters. The Code Editor uses some colors to differentiate categories of words or lines of text. The colors used are highly customizable. To change the colors, on the main menu, you can click Tools -> Options... In the Options dialog box, in the Environment section, click Fonts and Colors. To set the color of a category, in the Display Items section, click the category. In the Item Foreground combo box, select the desired color. If you want the words of the category to have a colored background, click the arrow of the Item Background combo box and select one:



In both cases, the combo boxes display a fixed list of colors. If you want more colors, you can click a Custom button to display the Color dialog box that allows you to "create" a color.



www.windev.com

Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



Fundamentals of Controls Methods

Introduction

A method is a procedure created as a member of a class. Methods are used to access or manipulate the characteristics of an object or a variable. There are mainly two categories of methods you will use in your classes:

- If you are using a control such as one of those provided by the Toolbox, you can call any of its public methods. The requirements of such a method depend on the class being used
- If none of the existing methods can perform your desired task, you can add a method to a class

[Windows Forms Controls](#)
A range of rich robust winforms controls to enrich your .net apps
www.divelements.co.uk

[Gold Exploration Company](#)
Over 35 Gold & Base Metal Projects Controls 2 Inferred Resources
www.eagleplains.com



Ads by Google

Control's Construction and Destruction

As you should know already, every class has a fundamental method called a default constructor. Every control of the .NET Framework is based on a class that has a default constructor and most of those classes have only one constructor: the default. The default constructor allows you to instantiate the class without necessarily initializing it. To use it, you must know the name of the control you want to use since each control bears the same name as its class. Here is an example:

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class WinControls
        Inherits Form

        Private btnReset As Button

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            btnReset = New Button()

        End Sub

        Public Shared Function Main() As Integer

            Application.Run(New WinControls())
            Return 0

        End Function

    End Class

End Module
```

If you are not planning to use a control straight from the .NET Framework, you can also create your own class that is derived from the class of the control, as we have mentioned in previous lessons.

As mentioned in the previous lesson, after instantiating a control, it is available but the user cannot see. Each control that acts as a parent of another control has a property called **Controls**. This property, which is a **ControlCollection** type, is equipped with an **Add()** method. If you want to display the new control to the user, you should pass it to the

Control.Controls.Add() method. Here is an example:

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class WinControls
        Inherits Form

        Private btnReset As Button

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            btnReset = New Button()

            Controls.Add(btnReset)
        End Sub

        Public Shared Function Main() As Integer

            Application.Run(New WinControls())
            Return 0

        End Function

    End Class

End Module
```

This displays the control to the user.

After using a control, it must be destroyed. Another detail of Windows controls is that they use or consume computer resources during their lifetime. When the controls are not used anymore, such as when their application closes, these resources should be freed and given back to the operating system to make them available to other controls. This task can be performed using the **Dispose()** method to the **Control** class, which can then be overridden by its child controls. The syntax of the **Control.Dispose()** method is:

```
Protected Overrides Sub Dispose(disposing As Boolean)
```

This method takes one argument, *disposing*, that indicates how the resources would be released. If this argument is passed with a **False** value, only the unmanaged resources would be released. If it is passed as **True**, then both managed and unmanaged resources would be released.

Creating New Methods

The Windows controls available from the .NET Framework and that we will use in our lessons. They are equipped with various methods ready to be used. Of course, no library can surely provide every single type of method that every programmer would use. For this reason, it will not be unusual that you need a method that is not available for a control you are using. In the same way, when you create a Windows Application that is based on a **Form** class, you will likely need a method that is not defined in the **Form** class. In this case, you can create your own and new method.

A method is created like a normal procedure. If you want to add it to a form, you can open the Code Editor and write your procedure outside of any existing procedure. Here is an example:

```
Imports System
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class WinControls
        Inherits Form

        Private btnReset As Button

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

    End Class

End Module
```

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google

```
Public Sub InitializeComponent()  
    btnReset = New Button()  
    btnReset.Text = "Reset"  
    btnReset.Location = New Point(20, 20)  
  
    Controls.Add(btnReset)  
End Sub  
  
Private Function CalculateRectangleArea(ByVal Recto As Rectangle) As Double  
    Return Recto.Width * Recto.Height  
End Function  
  
Public Shared Function Main() As Integer  
  
    Application.Run(New WinControls())  
    Return 0  
  
End Function  
  
End Class  
  
End Module
```

In the same way, if you derive a class from one of the existing classes because you want to get a custom control from it, you can declare a new method as you see fit and use it appropriately.

Probably the best way is to let the Code Editor insert the new method based on your specifications. To do that, in the Class View, first expand the name of the project. Then, right-click the name of the class where you want to add a new method, position the mouse on Add, and click Add Method. This would open the C# Method Wizard dialog box you can fill out and click Finish. After the method's body has been defined you

Microsoft Visual Basic Functions

Among all the languages of the .NET Framework and Microsoft Visual Studio, Visual Basic has the largest and the most impressive library of functions. There are so many of these functions, we cannot review them here.

Are your ISO standards current?



[Home](#)

Copyright © 2008 FunctionX, Inc.



Dynamic Control Creation

Introduction

The objects used in a Windows application are defined in various assemblies. To add one of these controls to your application, you must first know the name of its class. With this information, you can declare a variable of its class. For example, a command button is an object of type **Button** that is based on the **Button** class. The **Button** class is defined in the **System.Windows.Forms** namespace of the **System.Windows.Forms.dll** assembly. Based on this, to create a button, you can create a variable of type **Button**. Here is an example:

[Windows Forms Controls](#)

A range of rich robust winforms controls to enrich your .net apps

www.divelements.co.uk

[Motor Control Tutorials](#)

Free Web Tutorials from Galil, The World Leader in Motor Control.

www.Galilmc.com

[GT Group](#)

Global Manufacturers of Exhaust Brakes & EGR's for Diesel Engines

www.gtpp.co.uk

[ACN Nuclear Medicine](#)

BMD devices, Dexa devices for osteoporosis and nuclear medicine

www.acn.it



Ads by Google

```
Imports System
Imports System.Windows.Forms
```

```
Module Exercise
```

```
    Public Class Exercise
        Inherits Form
```

```
        Private btnSubmit As Button
```

```
        Public Sub New()
```

```
        End Sub
```

```
        Public Shared Function Main() As Integer
```

```
            Application.Run(New Exercise())
            Return 0
```

```
        End Function
```

```
    End Class
```

```
End Module
```

After declaring the variable, you can use the **New** operator to allocate memory for it:

```
Public Sub New()
    btnSubmit = New Button()
```

```
End Sub
```

This is also referred to as dynamically creating a control. After declaring the variable and allocating memory for it, the control is available but does not have a host, which makes it invisible. A control must be positioned on a container, like a form. The **Form** class itself contains a member variable named **Controls**. This member holds a list of the objects that are placed on the form. To specify that a control you have instantiated must be positioned on a form, the **Controls** member has a method named **Add**. Therefore, to make an object part of the form, pass its variable to the **Add()** method. Here is an example:

```
Imports System
Imports System.Windows.Forms
```

```
Module Exercise
```

```
    Public Class Exercise
        Inherits Form
```

```
        Private btnSubmit As Button
```

```
        Public Sub New()
```

```
            btnSubmit = New Button()
            Controls.Add(btnSubmit)
```

```

End Sub

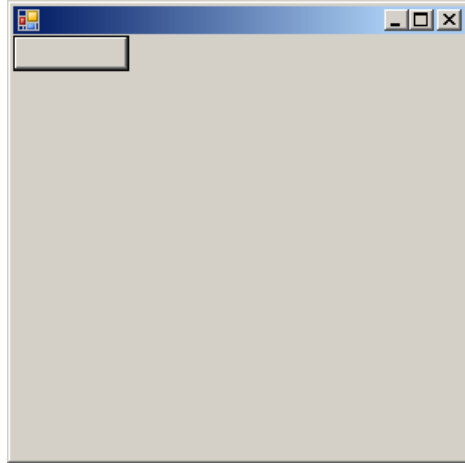
Public Shared Function Main() As Integer

    Application.Run(New Exercise())
    Return 0

End Function
End Class
End Module

```

This makes it possible for the control to appear on the form when the form displays to the user:



The two techniques of visual addition of objects and dynamic creation are the most used to add Windows controls to an application. The Windows controls are also called components.

Initializing the Components

Because there can be many controls used in a program, instead of using the constructor to initialize them, the Visual Studio standards recommend that you create a sub procedure called **InitializeComponent** to initialize the various objects used in your application. Then simply call that method from the constructor of your form. This would be done as follows:

```

Imports System
Imports System.Windows.Forms

Module Exercise

    Public Class Exercise
        Inherits Form

        Private btnSubmit As Button

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()

            btnSubmit = New Button()
            Controls.Add(btnSubmit)

        End Sub

        Public Shared Function Main() As Integer

            Application.Run(New Exercise())
            Return 0

        End Function
    End Class
End Module

```

Notice that the control is created in the **InitializeComponent()** method.

Using a Partial Class

Starting in Microsoft Visual Basic 2005, and probably getting close to C++, you can use two files to create and use a form. Each file would hold a partial definition of the class. As done in a

header file of a C++ application, the first file in VBasic would hold the variable or control declarations. While in C++ a header file holds the same name (but different extensions) as its corresponding source file, because VBasic does not have the concepts of header and source file, each file must have a different name. In Microsoft Visual Basic, the name of the first file of a form starts with the name of the form, followed by a period, followed by **Designer**, followed by a period, and followed by the vb extension.

Components Tracking on an Application

As you add and remove components on an application, you need a way to count them to keep track of what components, and how many of them, your application is using. To assist you with this, the .NET Framework provides a class named **Container**. This class is defined in the **ComponentModel** namespace that is itself part of the **System** namespace. To use a variable of this class in your application, declare a variable of type **Container**. Because no other part of the application is interested in this variable, you should declare it private. This can be done as follows:

```
Imports System
Imports System.Windows.Forms

Module Exercise

    Public Class Exercise
        Partial Public Class Exercise
            Inherits Form

            Private btnSubmit As Button

            Dim components As System.ComponentModel.Container

            Public Sub New()
                InitializeComponent()
            End Sub

            Public Sub InitializeComponent()
                btnSubmit = New Button()

                Controls.Add(btnSubmit)
            End Sub

        End Class

        Public Shared Function Main() As Integer

            Application.Run(New Exercise())
            Return 0

        End Function

    End Class
End Module
```

Ads by Google



[Motor Control Tutorials](#)

Free Web Tutorials from Galil, The World Leader in Motor Control.
www.Galilmc.com

[ACN Nuclear Medicine](#)

BMD devices, Dexa devices for osteoporosis and nuclear medicine
www.acn.it

[Custom Magnets](#)

Search Thousands of Catalogs for Custom Magnets
www.globalspec.com

[Optimum Controls](#)

Leading Technologies for Control. Cost-Effective Industrial Valves.
www.optimumcontrols.com

After this declaration, the compiler can keep track of the components that are part of the form.

Control Derivation

If you are using a .NET Framework control, you must know the name of the class on which the control is based (and each control is based on a particular class). If you have examined the types of classes available but none implements the behavior you need, you can first locate one that is close to the behavior you are looking for, then use it as a base to derive a new class.

To derive a class from an existing control, you can use your knowledge of inheritance. Here is an example:

```
Public Class Numeric
    Inherits System.Windows.Forms.TextBox

End Class
```

If you want to perform some early initialization to customize your new control, you can declare a constructor. Here is an example:

```
Public Class Numeric
    Inherits System.Windows.Forms.TextBox

    Public Sub New()

    End Sub

End Class
```

[Free SQL Server Tips](#)

MS SQL Server Tips & Techniques Solve SQL problems fast and free
www.mssqltips.com

Besides the constructor, in your class, you can add the fields and methods as you see fit. You can also use it to globally set a value for a variable of the parent class. Once the control is ready, you can dynamically use it like any other control. Here is an example:

```
Imports System
Imports System.Windows.Forms

Module Exercise

    Public Class Numeric
        Inherits System.Windows.Forms.TextBox

        Public Sub New()

        End Sub
    End Class

    Public Class Exercise
        Partial Public Class Exercise
            Inherits Form

            Private btnSubmit As Numeric

            Dim components As System.ComponentModel.Container

            Public Sub New()
                InitializeComponent()
            End Sub

            Public Sub InitializeComponent()
                btnSubmit = New Numeric()

                Controls.Add(btnSubmit)
            End Sub

        End Class

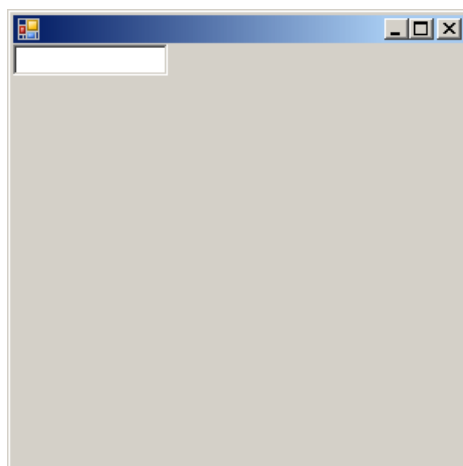
        Public Shared Function Main() As Integer

            Application.Run(New Exercise())
            Return 0

        End Function

    End Class
End Module
```

This produce:



[Home](#)

Copyright © 2008 FunctionX, Inc.

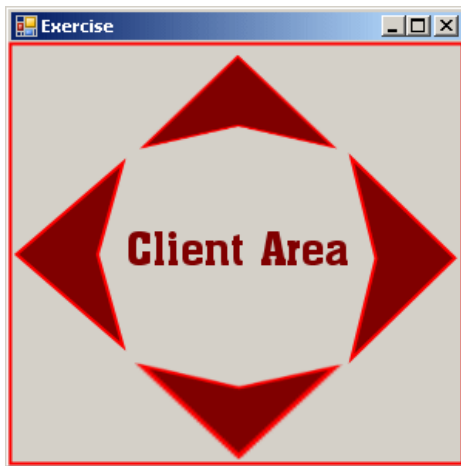
[Home](#)



Graphical Applications Fundamentals

The Client Area

On a form, the client area is the body of the form without the title bar, its borders and other sections we have not mentioned yet such as the menu, scroll bars, etc:



Server Applications

Get Updates About AMD Partnerships, Upcoming Events & More. Get Info.

www.amd.com/businessolutions

application recognition

Identify Applications with Nbar. Free Trial Version. Download Now !

www.netflowanalyzer.com/nbar

Free Mashup Composer

Taking Web 2.0 Apps to the Next Level w/ Mashups - Try It Free!

Serena.com/Mashups



Ads by Google

Ads by Google



Optimize Your Grid

Develop, Run and Manage Your Grid Using Platform Computing Products

www.platform.com/Products

Free Mashup Composer

Optimize Web 2.0 App Productivity w/ Mashups - Try our Free Mashups

Serena.com/Mashups

Replacement Pipe Saddles

Available 24x7, Replacement Pipe Saddles for all your needs

www.pipingtech.com/products

Visual Basic Code Library

Open Source Code Snippet Library. Free Community for Developers.

www.daniweb.com/code

Free Movie Preview API

100000 videos with metadata to make widgets, gadgets and applications

InternetVideoArchive.com

Besides the form, every control also has a client area. The role of the client area is to specify the bounding section where the control can be accessed by other controls positioned on it. Based on this, a control can be visible only within the client area of its parent. Not all controls can be parent.

Design and Run Times

Application programming primarily consists of adding objects to your project. Some of these objects are what the users of your [application](#) use to interact with the computer. As the [application developer](#), one of your jobs will consist of selecting the necessary objects, adding them to your application, and then configuring their behavior. There are various ways you can get a control into your application. If you are using Notepad or a [text editor](#) to add the objects, you can write code. If you are using Microsoft [Visual Basic](#), you can visually select an object and add it.

To create your applications, there are two settings you will be using. If a control is displaying on the screen and you are designing it, this is referred to as design time. This means that you have the ability to manipulate the control. You can visually set the control's appearance, its location, its size, and other necessary or available characteristics. The design view is usually the most used and the easiest because you can glance at a control, have a realistic display of it and configure its properties. The visual design is the technique that allows you to visually add a control and manipulate its display. This is the most common, the most regularly used, and the easiest technique.

The other technique you will be using to control a window is with code, writing the program. This is done by typing commands or instructions using the keyboard. This is considered, or referred to, as run time. This is the only way you can control an object's behavior while the user is interacting with the [computer](#) and your program.



[Home](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)

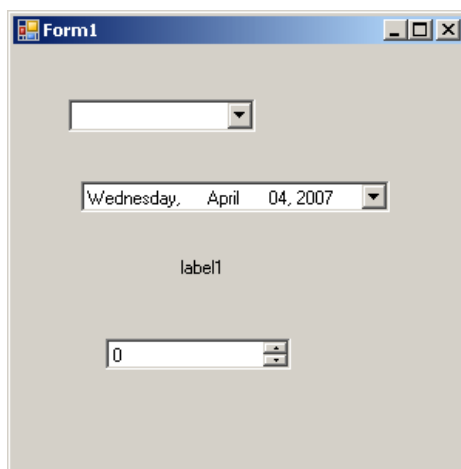


Application Design

Visual Control Addition

Introduction


To add a control to your application, you can select it from the Toolbox and click the desired area on the form. Once added, the control is positioned where your mouse landed. In the same way, you can add other controls as you judge them necessary for your application. Here is an example of a few controls added to a form:

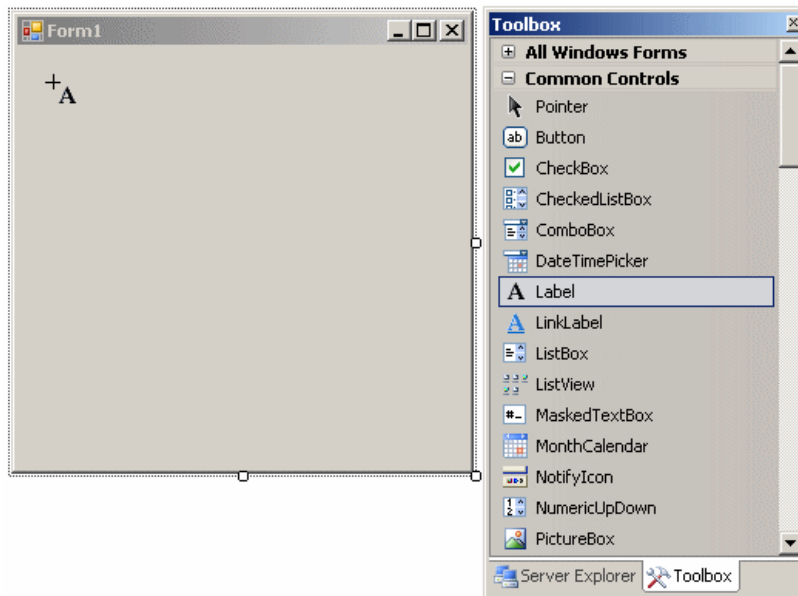






Alternatively, to add a control, you can also double-click it from the Toolbox and it would be added to the top-left section of the form.

If you want to add a certain control many times, before selecting it on the Toolbox, press and hold Ctrl. Then click it in the Toolbox. This permanently selects the control. Every time you click the form, the control would be added. Once you have added the desired number of this control, on the Toolbox, click the Pointer button to dismiss the control.

❖ Practical Learning: Using the Toolbox

1. Start Microsoft Visual Basic
2. To create a new application, on the main menu, click File -> New Project...
3. In the Templates list, click Windows Application
4. Set the Name to **DesignPractice1** and click OK
5. On the main menu, click View -> Toolbox.
Position the mouse on the Toolbox word and wait for the Toolbox to expand
6. Click the Label button  and position the mouse on the form

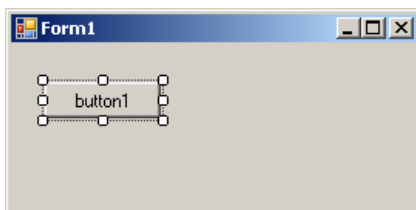


7. Click the form
8. Click the middle of the form to select it (the form)
9. To add another control, position the mouse again on the Toolbox word until the Toolbox has expanded
10. Find and double-click the TextBox button 
11. To use a hidden area of the form, position the mouse on the Toolbox word. When the Toolbox has expanded, click the Auto Hide button 
12. On the Toolbox, click the TreeView button  and click the left section of the form
13. After using the Toolbox, to hide it, click the Auto Hide button 
14. To execute the application, on the main menu, click Debug -> Start Without Debugging
15. After using it, close the form and return to your programming environment

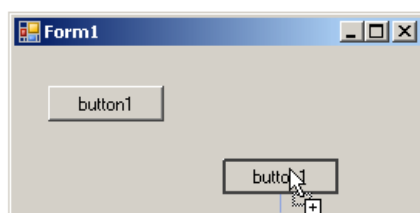
Copying a Control

We mentioned earlier how you could add a control many times. An alternative is to copy a control. To do this, on the form:

- Right-click the control and click Copy. Right-click another area of the form and click Paste
- Click (once) the control you want to copy



Press and hold Ctrl. Then drag the selected control to another area of the form. The mouse cursor would display a + plus indicating that the control is being duplicated:



Once you get to another area of the form, release the mouse and Ctrl

You can use these two techniques to copy a group of controls.

MANASHOSTING



MANASHOSTING

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >

www.manashosting.com
Ads by Google

**Doing the balancing act
with your current salary?**

SAVINGS



EXPENSES

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Application Design

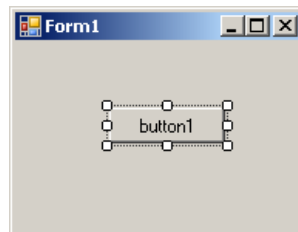
Selecting a Control

Introduction

When designing an application, you will manipulate the windows controls on a form. After adding a control to a form, before performing any operation on that control, you must first select it. You can also manipulate many controls at the same time. To do that, you will have to select all those controls.

Single Control Selection

To select one control on the form, you can simply click it. A control that is selected indicates this by displaying 8 small squares, also called handles, around it. Between these handles, the control is surrounded by dotted rectangles. In the following picture, the selected rectangle displays 8 small squares around its shape:

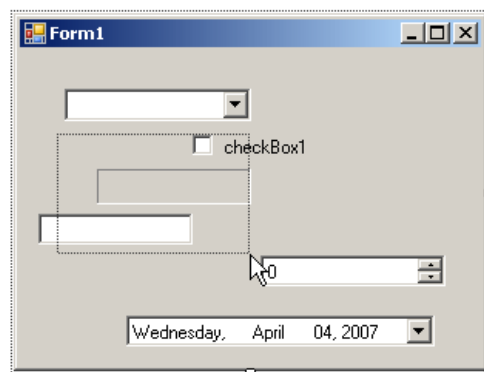


After selecting a control, you can manipulate it or change its characteristics, also called properties.

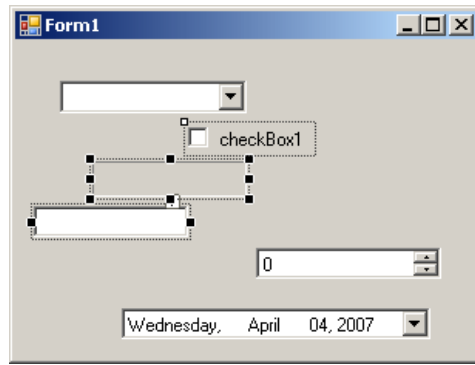
Multiple Control Selection

To select more than one control on the form, click the first. Press and hold either Shift or Ctrl. Then click each of the desired controls on the form. If you click a control that should not be selected, click it again. After selecting the group of controls, release either Shift or Ctrl that you were holding.

When a group of controls is selected, the last selected control displays 8 square handles around but its handles are white while the others are black. Another technique you can use to select various controls consists of clicking on an unoccupied area on the form, holding the mouse down, drawing a fake rectangle, and releasing the mouse:



Every control touched by the fake rectangle or included in it would be selected:



Control Deletion

If there is a control on your form but you don't need it, you can remove it from the application. To delete a control, first select it and then click or press Delete. You can also right-click a control and click Cut. To remove a group of controls, first select them, then click or press Delete or right-click the selection and click Cut.

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

**Only Rs.2000/-
Per Year**

[VIEW PLANS >](#)

MANASHOSTING



MANASHOSTING

www.manashosting.com
Ads by Google

Are your ISO standards current?



[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Application Design

Moving a Control

Introduction

When adding a control to a form, it assumes a position based on where the mouse landed when you clicked the form. Most of the time, that position will not be convenient. [Moving](#) a control consists of specifying its position by changing its previous left and top values. You can do this either graphically or programmatically.

To move a control graphically:

- Position the mouse on it until the [cursor](#) changes into a cross:



Then click and drag left, right, up or down, until you get to the desired location

[Delhi Movers/Packers](#)

We'll get you 5 free contacts of Delhi Movers & Packers. Enquire
Yellowpages.Sulekha.com/Delhi

[Windows Forms Controls](#)

A range of rich robust winforms controls to enrich your .net apps
www.divelements.co.uk



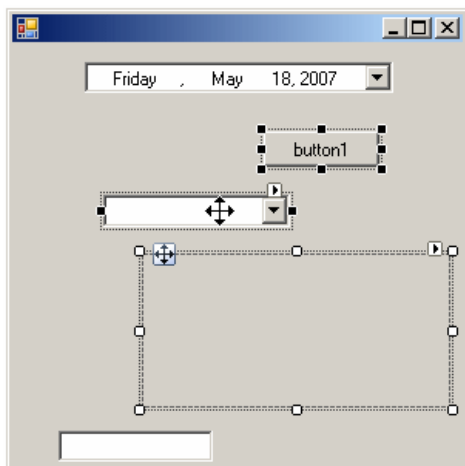
Ads by Google

- Click the control (once) to select it. Using your keyboard, press either the left, the up, the right, or the down arrow keys to move the control until you get the desired position

Moving Various Controls

You can also move various controls at the same time. To do this, first select the controls:

- Position the mouse on one of the selected controls:



Then click and drag left, right, up, down, or diagonally, until you get the desired position

- Press the left, the up, the right, or the down arrow keys to move the control until you get the desired position

Locking a Control

After adding a control to a form, you can move the control to change its position, as we will learn in the next few sections. In the next lesson, we will learn how you can change the size of a control. The availability of these two operations is controlled by a Boolean property named **Locked**. The [default value](#) of this property is **False**. Therefore, to prevent the control from being moved or resized, access its Properties window and set the **Locked** property to **True**.

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google

Unlimited Webspace and Unlimited Bandwidth

<ul style="list-style-type: none"> ⇒ 100 Email Ids of 10 GB ⇒ All Database FREE ⇒ 5000 Templates / Website Builder 	<p style="text-align: center; font-weight: bold; font-size: 1.2em; color: #000080;">FREE</p> <p style="text-align: center; font-weight: bold; color: #000080;">Domain Registration</p> <p style="text-align: center; font-weight: bold; color: #FFA500; font-size: 1.5em;">Only Rs.2000/-</p>
<ul style="list-style-type: none"> ⇒ Full Control Panel ⇒ 200 SEO Tools/Softwares/Weblinks 	

www.manashosting.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)

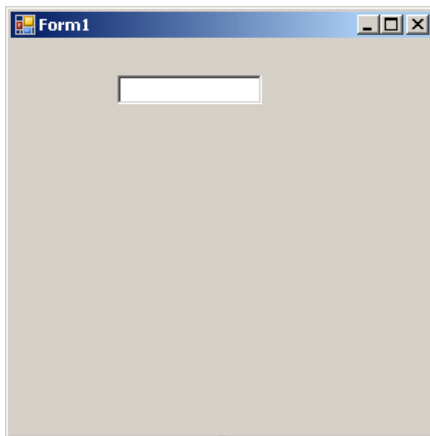


Application Design

Control Alignment

Introduction

Microsoft Visual Studio 2005 provides various tools to assist you with aligning your controls on a form. You can first add a control to a form and position the control the way you want. Here is an example:



[Windows Forms Controls](#)

A range of rich robust winforms controls to enrich your .net apps
www.divelements.co.uk

[Free Datagrid for WPF](#)

100% stylable and templatable, with rich in-place editing & more
xceed.com/Grid_WPF_Intro.html

[Laminar Air Flow](#)

High quality Laminar Air Flow, Low Cost, Call Now
www.ChemPharmIndia.com

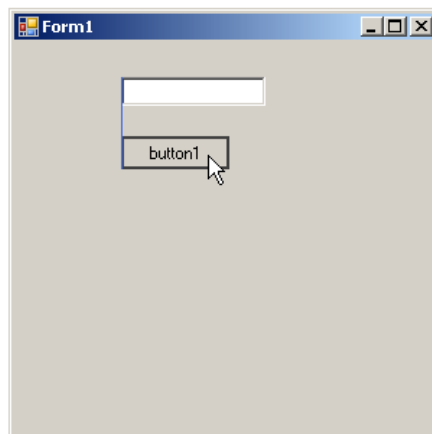
[Holiday Cottages Scotland](#)

large selection traditional holiday cottages in the Scottish Borders
www.unique-cottages.co.uk



Ads by Google


Once you have a control on your form, you can add another control as we saw in the previous lesson. To position the other control, you can use the previous one as a reference. To assist you with this, when moving the new control to position it, a guiding vertical line would show you the alignment to follow with regards to an existing control. Here is an example:

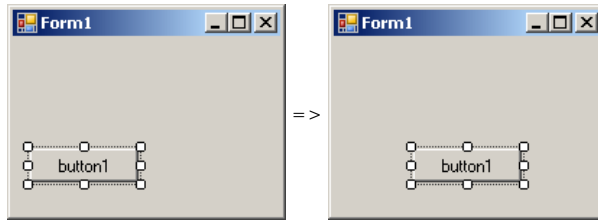


Using this approach, once the control is aligned fine, you can release the mouse. As another technique, after positioning one or a few controls, to align a control with reference to another, press and hold Ctrl. Then press the left, the up, the right, or the down arrow key. When you press one of these keys, the control would move to align itself with the next control in that direction. Once the alignment is to your liking, release Ctrl.

There are various other techniques you can use to align the controls. We will review them.




Control Centering Towards the Center of the Form

If you have a certain control on the form and want to position it exactly at equal distance between the left and the right borders of the form, select the control, then click the Center Horizontally button on the Layout toolbar :






Horizontal Alignment

Horizontal alignment affects controls whose distance from the left border of the parent must be the same. To perform this type of alignment, the Layout toolbar provides the necessary buttons. The same actions can be performed using menu items of the Format group on the main menu. The options are as follows:

Button	Name	Format Menu	Description
	Align Lefts	Align -> Lefts	All selected controls will have their left border coincide with the left border of the base control
	Align Centers	Align -> Centers	The middle handles of the selected controls will coincide with the middle handles of the base control
	Align Rights	Align -> Rights	All selected controls will have their right border coincide with the right border of the base control


Vertical Alignment

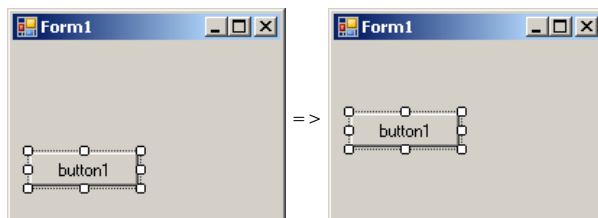
As seen above, the horizontal-oriented buttons allow moving controls left or right. Another option you have consists of moving controls up or down for better alignment. Once again you must first select the controls. Then on the Layout toolbar or the Format group of the main menu, use the following options:

Button	Name	Format Menu	Description
	Align Tops	Align -> Tops	All selected controls will have their top border coincide with the top border of the base control but their left border would have the same distance with the left border of the parent
	Align Middles	Align -> Middles	The top handles of the selected controls will align vertically with the top handle of the base control
	Align Bottoms	Align -> Bottoms	All selected controls will have their bottom border coincide with the bottom border of the base control but their left border would have the same distance with the left border of the parent

Another valuable option you have consists of controlling the alignment of objects with regards to the extreme borders of controls of the selected group.

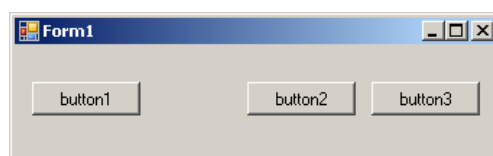
Control Centering Towards the Middle of the Form

You can also position one or more controls in the middle of the form. To do that, select the control, then click the Center Vertically button on the Layout toolbar .




Horizontal Spacing and Alignment

Suppose you have a group of horizontally aligned controls as follows:

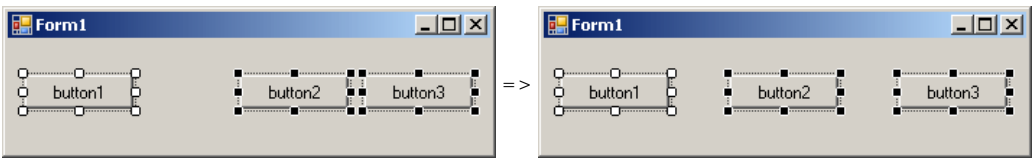



Obviously the buttons on this form are not enjoying the most professional alignment. For one thing, the distance between the Continue and the Submit buttons is longer than the distance between the Submit and the Deny buttons. The Layout toolbar and the Format group of the main menu allow you to specify a better horizontal alignment of controls with regards to each other. The options available are:

Button	Name	Format
	Make Horizontal Spacing Equal	Horizontal Spacing -> Make Equal

Result: The Forms Designer will calculate the horizontal distances that separate each combination of two controls and find their average. This average is applied to the horizontal distance of each combination of two controls:

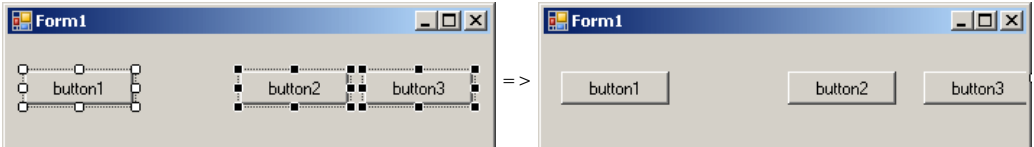
The left control is used as reference



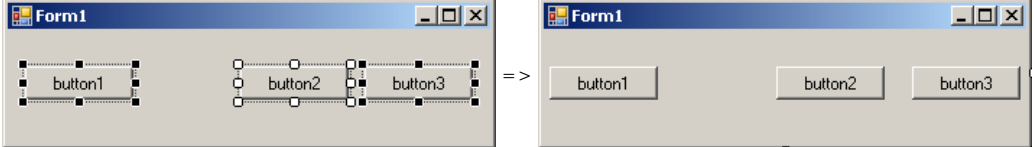
Button	Name	Format
	Increase Horizontal Spacing	Horizontal Spacing -> Increase

Result: The Forms Designer will move each control horizontally, except the base control (the control that has white squares) by one unit away from the base control. This will be done every time you click the Increase Horizontal Spacing button or the Format -> Horizontal Spacing -> Increase menu item:

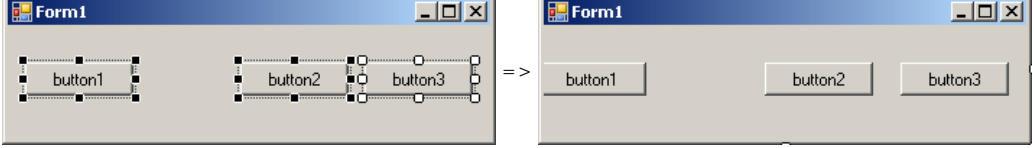
The left control is used as reference




The middle control is used as reference



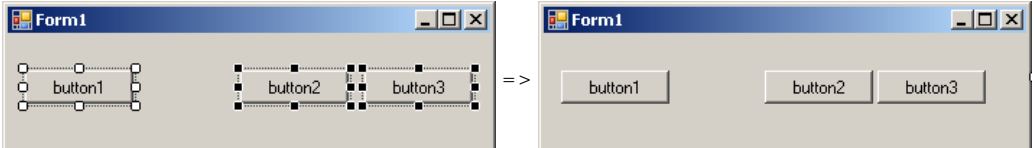
The right control is used as reference



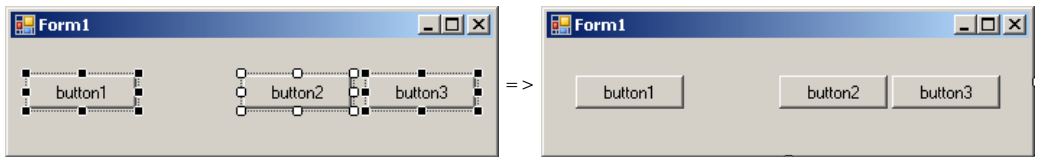
Button	Name	Format
	Decrease Horizontal Spacing	Horizontal Spacing -> Decrease

Result: The Forms Designer will move each control horizontally, except the base control (the control that has darker handles) by one unit towards the base control. This will be done every time you click the Decrease Horizontal Spacing button or the Format -> Horizontal Spacing -> Decrease menu item:

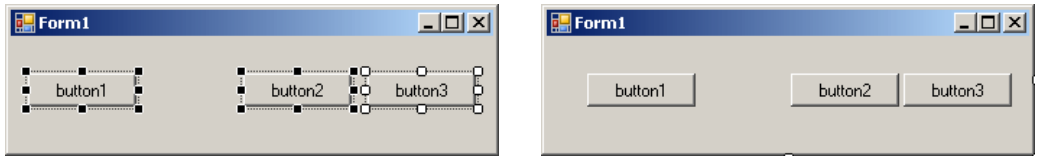
The left control is used as reference



The middle control is used as reference



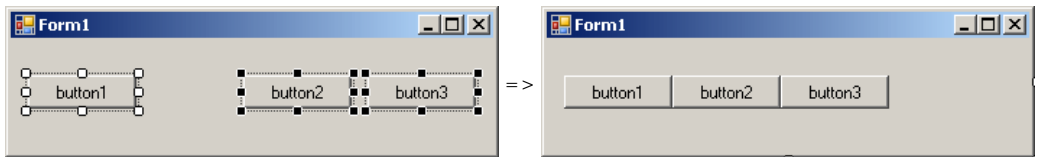
The right control is used as reference



Button	Name	Format
	Remove Horizontal Spacing	Horizontal Spacing -> Remove

Result: The Forms Designer will move all controls (horizontally), except for the left control, to the left so that the left border of a control touches the right border of the next control:

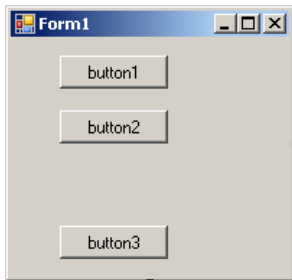
The left control is used as reference



Ads by Google **V V**
Integrated Access Control
 Biometric, RFID, TCP/IP panel Proximity, MiFare & CCTV
www.amtel-security.com

Vertical Spacing and Alignment

Suppose you have a group of horizontally positioned controls as follows:

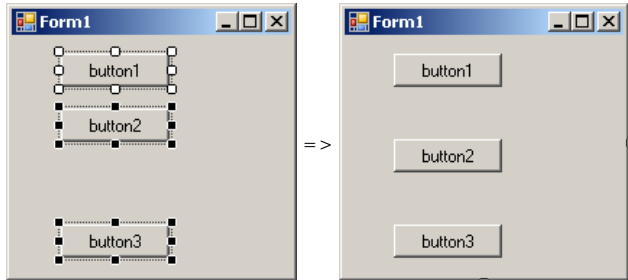


The buttons on this form are not professionally aligned with regards to each other. Once again, the Layout toolbar and the Format group of the main menu allow you to specify a better vertical alignment of controls relative to each other. The options available are:

Button	Name	Format
	Make Vertical Spacing Equal	Vertical Spacing -> Make Equal

Result: The Forms Designer will calculate the total vertical distances that separate each combination of two controls and find their average. This average is applied to the vertical distance of each combination of two controls:

The top control is used as reference



Button	Name	Format
	Increase Vertical Spacing	Vertical Spacing -> Increase

Result: The Forms Designer will move each control vertically, except the base control (the control that has darker handles) by one unit away from the base control. This will be done

GT Group
 Global Manufacturers of Exhaust Brakes & EGR's for Diesel Engines
www.gtpp.co.uk

ACN Nuclear Medicine
 BMD devices, DEXA devices for osteoporosis and nuclear medicine
www.acn.it

JavaScript Menu Designs
 Powerful, easy, truly cross-browser Buy AllWebMenus, save time & money!
www.liqno.com

Motor Control Tutorials
 Free Web Tutorials from Galil, The World Leader in Motor Control.
www.Galilmc.com

every time you click the Increase Horizontal Spacing button or the Format -> Horizontal Spacing -> Increase menu item:

The top control is used as reference

The middle control is used as reference

The bottom control is used as reference

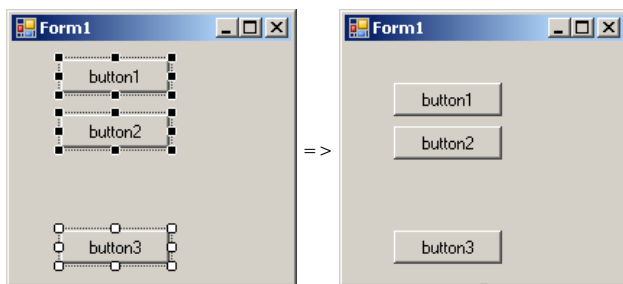
Button	Name	Format
	Decrease Vertical Spacing	Vertical Spacing -> Decrease


Result: The Forms Designer will move each control, except the base control (the control that has darker handles) by one unit towards the base control. This will be done every time you click the Decrease Horizontal Spacing button or the Format -> Horizontal Spacing -> Decrease menu item:

The top control is used as reference

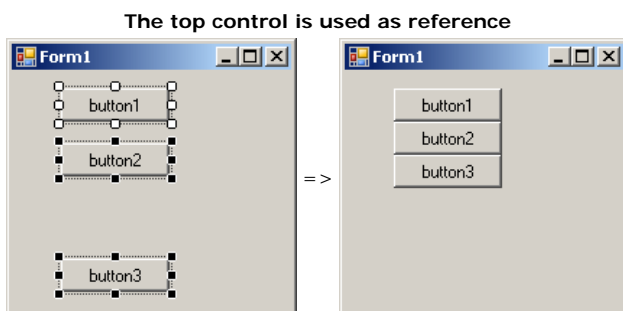
The middle control is used as reference

The bottom control is used as reference



Button	Name	Format
	Remove Vertical Spacing	Vertical Spacing - > Remove

Result: The Forms Designer will move all controls vertically, except for the top control, to the top so that the top border of a control touches the bottom border of the next control towards the top:



Are your ISO standards current?





Application Design

Resizing the Controls

Introduction

All graphical controls, including the form, can be resized using guiding mouse cursors or the keyboard. To resize a control, first select it. Except for the form, whenever a control is selected, there are eight handles around it. To resize the control, position your mouse on one of the handles. The mouse pointer will change, indicating in what direction you can move to resize the control.

Cursor	Role
	Moves the seized border in the North-West <-> South-East direction
	Shrinks or heightens the control
	Moves the seized border in the North-East <-> South-West direction
	Narrows or enlarges the control

Before resizing a control, as mentioned already, first select it. To enlarge a control:

- Position the mouse on the right (or the left) handle. Then click and drag in right (or left) direction. Once you get the desired width, release the mouse
- Press and hold Shift. Then press the right arrow key as many times as you want. Once you get the desired width, release Shift

To narrow a control:

- Position the mouse on its right (or its left) handle. Then click and drag in the left (or the right) direction. Once you get the desired width, release the mouse
- Press and hold Shift. Then press the left arrow key as many times as you want. Once you get the desired width, release Shift

To heighten a control:

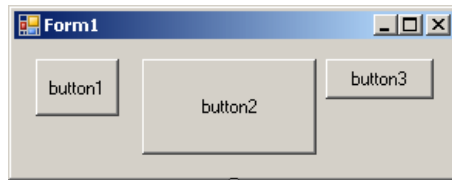
- Position the mouse on its top (or its bottom) handle. Then click and drag in the top (or the bottom) direction. Once you get the desired width, release the mouse
- Press and hold Shift. Then press the up arrow key as many times as you want. Once you get the desired width, release Shift

To shrink a control:

- Position the mouse on its top (or its bottom) handle. Then click and drag in the bottom (or the top) direction. Once you get the desired width, release the mouse
- Press and hold Shift. Then press the down arrow key as many times as you want. Once you get the desired width, release Shift

The Width and Height of a Control

Imagine you have added three controls to your form and, after spending some time designing them, they appear as follows:



The dimensions of the controls are not set professionally. As seen above, you can resize by dragging their borders but this might take a while if you want them to have the same width, the same height, or both the same height and width. The dimensions of a control or a group of controls are carried by a **Size** value.

At design time, to change the dimensions of a control, first click it. Then, in the Properties window, change the values of its **Size** property.

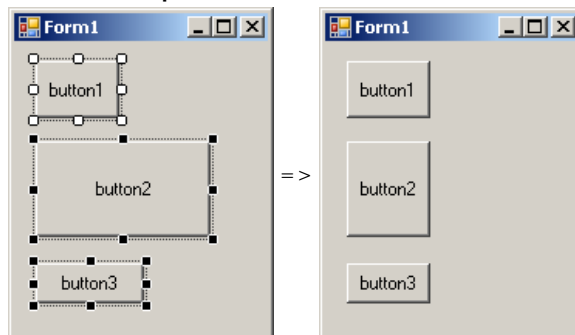
To change the dimensions of a group of controls, first select them. Then, in the Properties window, change the values of the **Size** field. The new value would be applied to all selected controls. Alternatively, the Form Designer provides tools to automatically do this for you.

To synchronize the widths of a group of controls, first select them. Then, on the Layout toolbar or on the Format group of the main menu, select:

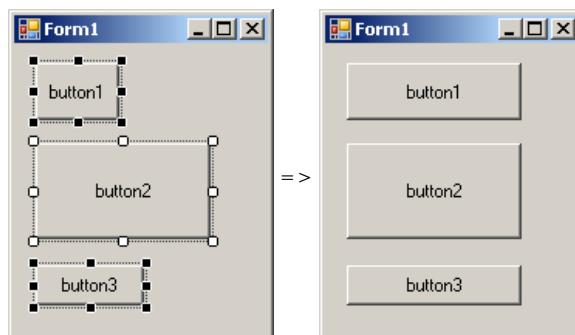
Button	Name	Format
	Make Same Width	Make Same Size -> Width

Result: All controls, except for the base control (the control that has the dark handles), will be resized horizontally so they have the same width as the base control:

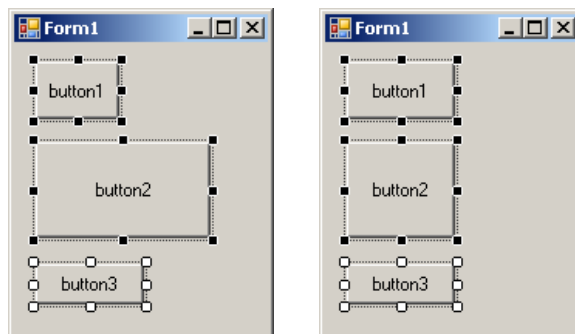
The top control is used as reference



The middle control is used as reference




The bottom control is used as reference



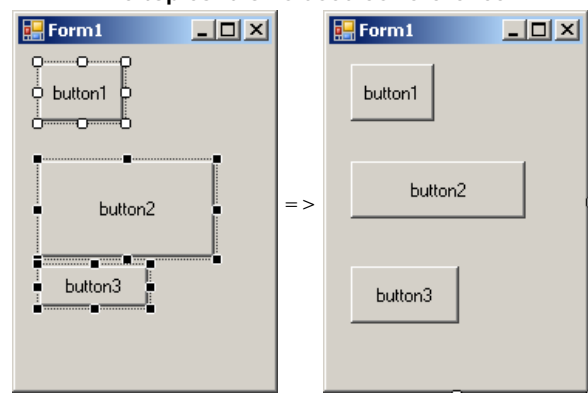
To set the same height to a group of controls, first select them. Then, on the Layout toolbar or on the Format group of the main menu, select:



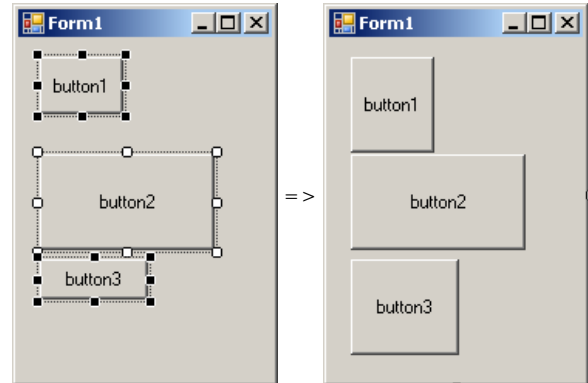
Button	Name	Format
	Make Same Height	Make Same Size -> Height

Result: All controls, except for the base control (the control that has the dark handles), will be resized vertically so they have the same height as the base control:

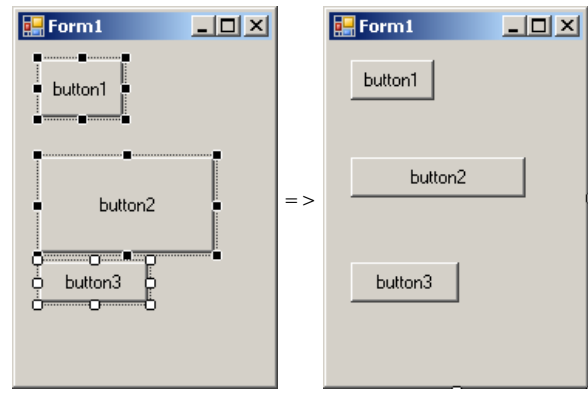
The top control is used as reference



The middle control is used as reference



The bottom control is used as reference



MANASHOSTING



MANASHOSTING

Unlimited Pack


- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

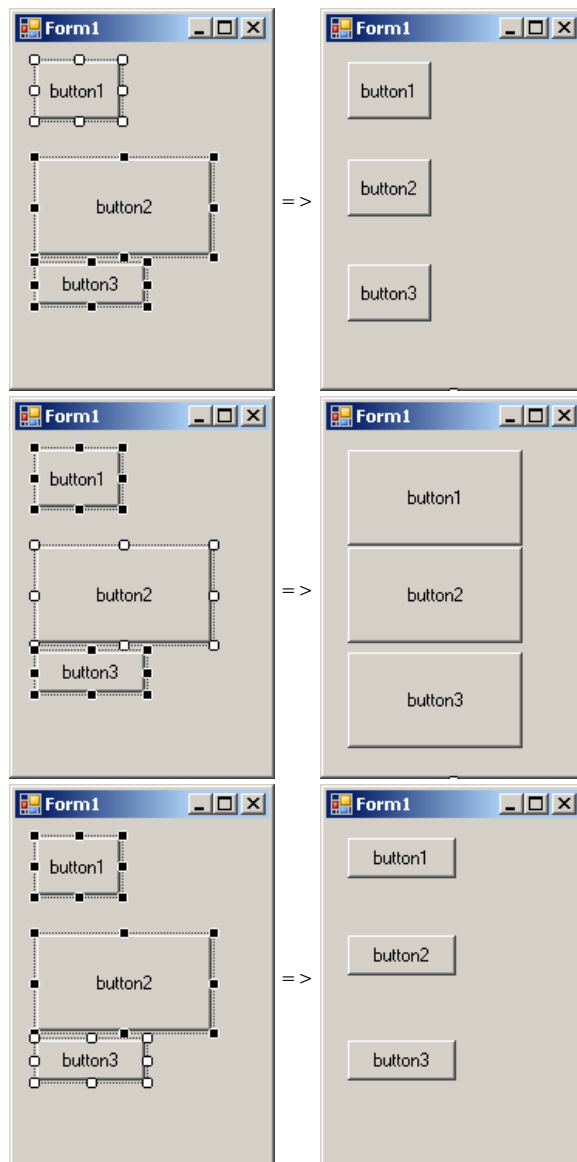
VIEW PLANS >

www.manashosting.com
Ads by Googale

To set the same width and the same height to a group of controls, first select them. Then, on the Layout toolbar or on the Format group of the main menu, select:

Button	Name	Format
	Make Same Size	Make Same Size -> Both

Result: The Form Designer will calculate the sum of the heights of all controls and find their average height (*AvgHeight*). It will also calculate the sum of the widths of all controls and find their average width (*AvgWidth*). These averages will be applied to the height and the width respectively of each control:

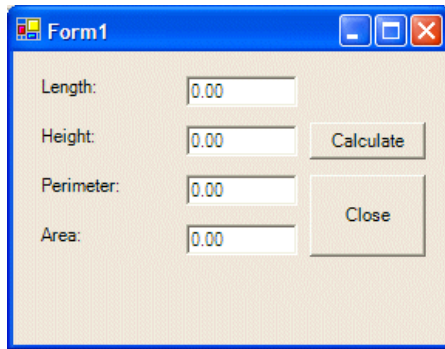


❖ Practical Learning: Setting the Locations and Sizes of Controls

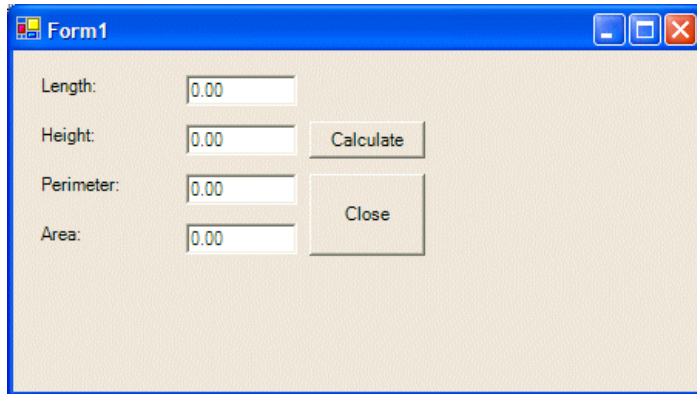
1. To create a new application, on the main menu, click File -> New -> Project...
2. In the Templates list, click Windows Application
3. In the Name box, replace the content with **Rectangle2** and press Enter
4. From the Toolbox and from what we learned in the previous lesson, add four labels, four text boxes, and two buttons to the form
5. Based on what we have reviewed so far, design the form as follows:

6. To test the application, on the Standard toolbar, click the Start Without Debugging button





7. While the form is displaying, drag its right border to widen it. Also drag its bottom border to heighten it



8. Close the form and return to your programming environment
9. Save all

Unlimited Webspace and Unlimited Bandwidth **FREE** **Only Rs. 2000/-**
Domain Registration

⇒ 100 Email Ids of 10 GB ⇒ Full Control Panel
⇒ All Database FREE ⇒ 200 SEO Tools/Softwares/Weblinks
⇒ 5000 Templates / Website Builder

www.manashosting.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



Introduction to Applications Menus

The Main Menu

Introduction

When it comes to a [restaurant](#), a menu is a list of food items that the business offers to its customers. For a computer [application](#), a menu is a list of actions that can be performed on that program. To be aware of these actions, the list must be presented to the user upon request. On a typical DOS application, a menu is presented with numerical or character options that the user can select from. An example would be:

Here are the various options:

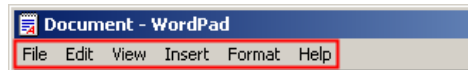
1. Register a new student
2. Review a student's information
3. Enter student's grades
4. Close the application



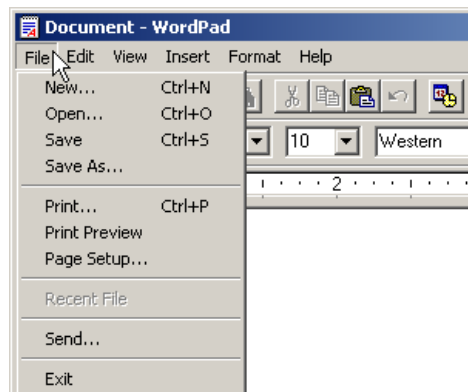
The user would then enter the number (or character) that corresponds to the desired option and continue using the program. For a graphical application, a menu is presented as a list of words and, using a mouse or a keyboard, the user can select the desired item from the menu.

To enhance the functionality of a graphical application, also to take advantage of the mouse and the keyboard, there are various types of menus. A menu is considered a main menu when it carries most of the actions the user can perform on a particular application. Such a menu is positioned in the top section of the form in which it is used. >

A main menu is divided in categories of items and each category is represented by a word. In WordPad, the categories of menus are File, Edit, View, Insert, Format, and Help:



To use a menu, the user first clicks one of the words that displays on top. When clicked, the menu expands and displays a list of items that belong to that category. Here is an example:




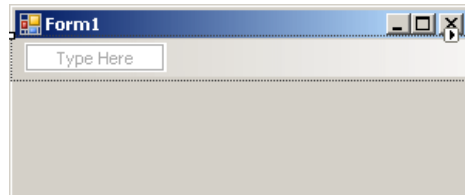
There is no strict rule on how a menu is organized. There are only suggestions. For example, actions that are related to file processing, such as creating a new file, opening an existing file, saving a file, printing the open file, or closing the file usually stay under a category called File. In the same way, actions related to viewing documents can be listed under a View menu category.

Main Menu Creation

To support actions that are used to graphically enhance the functionality of an application, the [.NET Framework](#) provides the **ToolStrip** class. To support menus for an application,

the .NET Framework provides the **MenuStrip** class (in Microsoft Visual Studio 2002 and 2003, the main menu was implemented through the **MainMenu** class, which is still available but lacks some features).

To graphically create a main menu, in the Menus & Toolbars section of the Toolbox, you can click the **MenuStrip** button  and click the form that will use the menu. After clicking the form, an empty menu is initiated:



Like every control, the main menu must have a name. After adding the menu strip to a form, you can accept the suggested name or, in the Properties window, click (Name) and type the desired name. You can then use the menu placeholder to add the necessary menu item(s)..

To programmatically create a main menu, declare a handle to **MenuStrip** class and initialize it with its default constructor. Because the main menu is primarily a control, you must add it to the list of controls of the form. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

        Private MenuMain As MenuStrip

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            MenuMain = New MenuStrip
            Controls.Add(MenuMain)
        End Sub

    End Class

    Function Main() As Integer

        Dim frmStart As Starter = New Starter

        Application.Run(frmStart)

        Return 0
    End Function

End Module
```

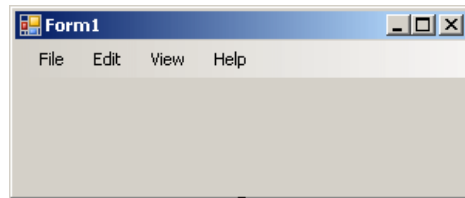
Menu Categories

In our introduction, we saw that a main menu was made of categories represented in the top section. After adding a MenuStrip, you can start creating the desired menu categories. To graphically create a menu category:

- In the menu strip, you can click the Type Here line and type the desired string
- In the menu strip, you can click Type Here. Then, in the Properties window, click the Text field and type the desired string

To create the next menu category, you can click Type Here on the right side of the previous menu category. In the same way, you can continue creating the desired menu categories.

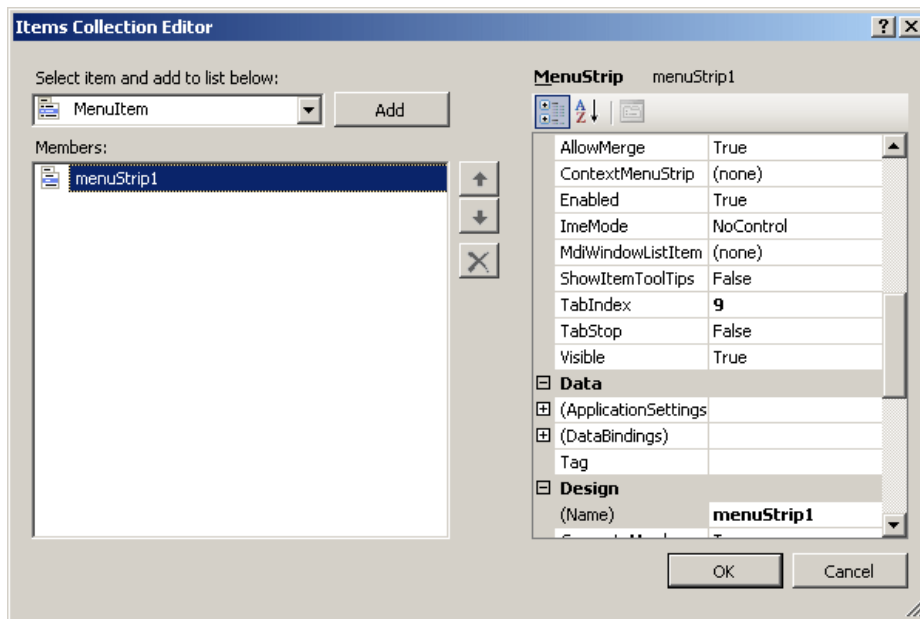
Here is an example:



Besides clicking Type Here and typing a string, an alternative is to get assisted by a dialog box. To open it:

- Under the form, you can click the menu strip object and, in the Properties window, you can click the ellipsis button of the Items field
- Under the form, you can right-click the menu strip and click Edit Items...

A dialog box, titled Items Collection Editor, would come up:



To create a menu category, in the Select Item And Add To List Below combo box, select MenuItem and click the Add button. In the right list, configure the menu item. At a minimum, you should specify its caption in the **Text** field. Like every control, each menu item has a name. To make sure you can easily recognize it in your code, when creating a menu item, you should give it a name unless you are contempt with the suggested one. After creating the menu categories, you can click OK and keep the dialog box opened for more options.

To support menu items, the .NET Framework provides the **ToolStripMenuItem** class. Using it, to create a menu category, declare a handle to this class and initialize it using one of its constructors. The default constructor is used to create a menu item without specifying any of its options. Here is an example:

Module [Exercise](#)

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuFile As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip
        MenuFile = New ToolStripMenuItem
        Controls.Add(MenuMain)
    End Sub

End Class
```

End Module

To specify the caption that will be displayed on a menu category, you can use the following constructor of the **ToolStripMenuItem** class:

```
Public Sub New(text As String)
```

Here is an example:

```
Public Sub InitializeComponent()
    MenuMain = New MenuStrip
    MenuFile = New ToolStripMenuItem("File")
    Controls.Add(MenuMain)
End Sub
```

If you had instantiated the class using its default constructor, to specify its caption, the **ToolStripMenuItem** class is equipped with the **Text** property. Therefore, assign a string to this property. Here is an example:

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuFile As ToolStripMenuItem
    Private MenuEdit As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip
        MenuFile = New ToolStripMenuItem("File")
        MenuEdit = New ToolStripMenuItem()
        MenuEdit.Text = "Edit"

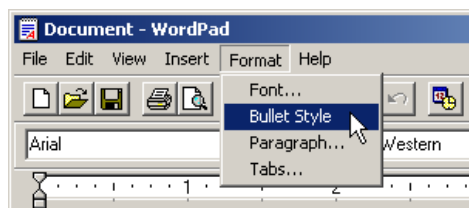
        Controls.Add(MenuMain)
    End Sub
End Class
```

End Module

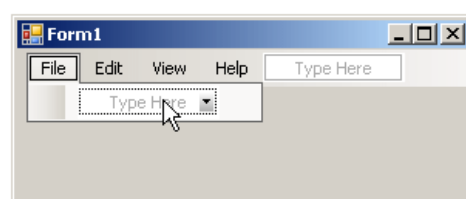
In the same way, you can create as many menu categories as you judge necessary for your application.

Introduction to Menu Items

In our introduction, we saw that if you click a menu category, a list comes up. Here is an example:



The objects under a menu category are referred to as menu items. To graphically create a menu item, first access the menu strip, which you can do by clicking it under the form. On the form, click a menu category. Once you do, a placeholder would be displayed under it:



To create a menu item:

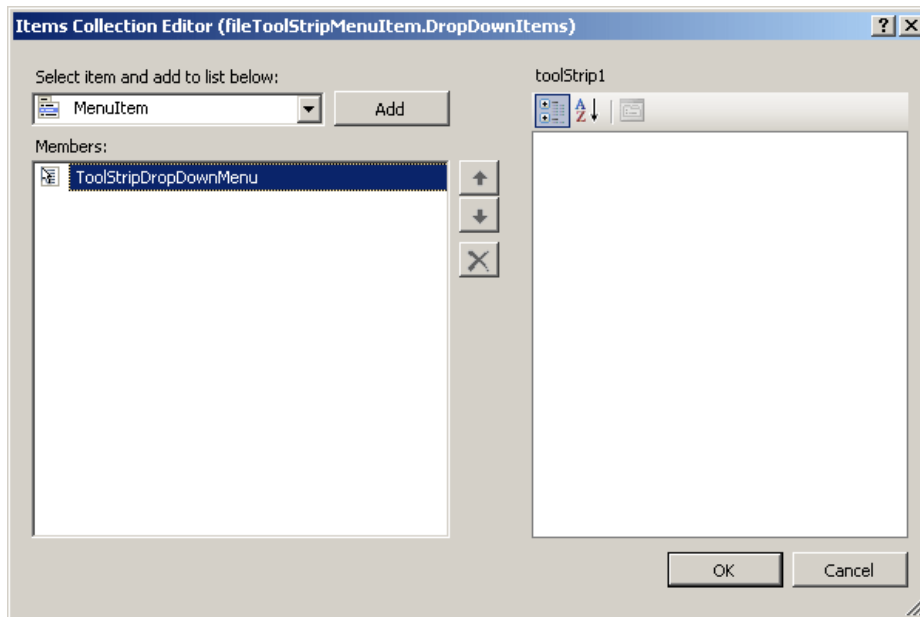
- Under a category, click Type Here and type the desired caption.
- Click the menu category. Then, in the Properties window, click Text and type the desired string.

In the same way, to create the next menu item, under the same category, click the next Type Here and type the desired caption or change the Text value in the Properties window.

An alternative is to use a dialog box. To access it, in the menu designer:

- Right-click the menu category and click Edit Drop Down Items...
- Click the menu category. Then, in the Properties window, click the ellipsis button of the DropDownItems field

The Items Collection Editor dialog box would come up:



To create a menu item, in the Select Item And Add To List Below combo box, select MenuItem and click Add. On the right side, configure the menu item as you see fit. At a minimum, you should specify its caption in the Text field.

Both the menu category and the menu item are created using the **ToolStripMenuItem** class. Here are examples:

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuFile As ToolStripMenuItem
    Private MenuFileNew As ToolStripMenuItem
    Private MenuFileExit As ToolStripMenuItem
    Private MenuEdit As ToolStripMenuItem
    Private MenuEditCopy As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip()
        MenuFile = New ToolStripMenuItem("File")
        MenuFileNew = New ToolStripMenuItem("New")
        MenuFileExit = New ToolStripMenuItem("Exit")
        MenuEdit = New ToolStripMenuItem("Edit")
        MenuEditCopy = New ToolStripMenuItem("Copy")
    End Sub
End Class
```

End Module

Associating Menu Items to Menu Categories

If you visually create your main menu, the form designer takes care of most details behind the scenes. For example, each menu item is automatically added to its parent menu category. If you programmatically create your main menu, you must associate each menu item to its parent menu category.

To support menu categories, **ToolStripMenuItem**, the class used to create menu categories, is derived from a class named **ToolStripDropDownItem**. The **ToolStripDropDownItem** class is abstract, which means you cannot instantiate it. Instead, it provides functionality to other classes derived from it. The **ToolStripDropDownItem** class is based on the **ToolStripItem** class.

To support menu items, the **ToolStripDropDownItem** class is equipped with a property named **DropDownItems**. This property is of type **ToolStripItemCollection**, which a collection-based class. The **ToolStripItemCollection** class implements the **IList** and the **ICollection** interfaces.

To specify that a menu item will be part of a menu category, call the **Add()** method of the **ToolStripItemCollection** class. This method is overloaded with various versions. One of the versions uses the following syntax:

```
Public Function Add(value As ToolStripItem) As Integer
```

This version allows you to pass a **ToolStripItem**-type of item class, such as a **ToolStripMenuItem** object. Here is an example:

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuEdit As ToolStripMenuItem
    Private MenuEditCopy As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip
        MenuEdit = New ToolStripMenuItem("Edit")
        MenuEditCopy = New ToolStripMenuItem("Copy")
        MenuEdit.DropDownItems.Add(MenuEditCopy)
    End Sub
End Class
```

End Module

The **ToolStripItemCollection** class also allows you to create a menu item without going through a **ToolStripItem**-type of object. To support this, it provides the following version of its **Add()** method:

```
Public Function Add(text As String) As ToolStripItem
```

This method takes as argument the text that the menu item would display and it returns the **ToolStripItem** item that was created. Here is an example:

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuEdit As ToolStripMenuItem
    Private MenuEditCopy As ToolStripMenuItem
    Private MenuEditPaste As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub
```

```

Public Sub InitializeComponent()
    MenuMain = New MenuStrip
    MenuEdit = New ToolStripMenuItem("Edit")
    MenuEditCopy = New ToolStripMenuItem("Copy")
    MenuEdit.DropDownItems.Add(MenuEditCopy)
    MenuEditPaste = CType(MenuEdit.DropDownItems.Add("Paste"), _
        ToolStripMenuItem)
End Sub

End Class

End Module

```

Instead of adding one menu item at a time, you can create an array of menu items and then add it to a category in one row. To support this, the **ToolStripItemCollection** class implements the **AddRange()** method. This method is overloaded with two versions. One of the versions uses the following syntax:

```
Public Sub AddRange(toolStripItems As ToolStripItem())
```

When calling this method, you must pass it an array of **ToolStripItem**-type of objects. Here are two examples:

Module Exercise

```

Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuFile As ToolStripMenuItem
    Private MenuFileNew As ToolStripMenuItem
    Private MenuFileOpen As ToolStripMenuItem
    Private MenuFileExit As ToolStripMenuItem
    Private MenuEdit As ToolStripMenuItem
    Private MenuEditCopy As ToolStripMenuItem
    Private MenuEditPaste As ToolStripMenuItem
    Private MenuHelp As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip

        MenuFile = New ToolStripMenuItem("File")
        MenuFileNew = New ToolStripMenuItem("New")
        MenuFileOpen = New ToolStripMenuItem("Open")
        MenuFileExit = New ToolStripMenuItem("Exit")
        Dim MenuFileItems() As ToolStripMenuItem = _
            {MenuFileNew, MenuFileOpen, MenuFileExit}
        MenuFile.DropDownItems.AddRange(MenuFileItems)

        MenuEdit = New ToolStripMenuItem("Edit")
        MenuEditCopy = New ToolStripMenuItem("Copy")
        MenuEdit.DropDownItems.Add(MenuEditCopy)
        MenuEditPaste = CType(MenuEdit.DropDownItems.Add("Paste"), _
            ToolStripMenuItem)

        MenuHelp = New ToolStripMenuItem("Help")
        Dim mnuHelpItems() As ToolStripMenuItem = _
            {
                New ToolStripMenuItem("Search"), _
                New ToolStripMenuItem("Contents"), _
                New ToolStripMenuItem("Index"), _
                New ToolStripMenuItem("Support Web Site"), _
                New ToolStripMenuItem("About this application") _
            }
        MenuHelp.DropDownItems.AddRange(mnuHelpItems)
    End Sub

End Class

End Module

```

Associating Menu Categories to the Main Menu

If you visually create your main menu, each menu category is automatically assigned to the menu strip. If you programmatically create your main menu, you must take care of this in

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google

order to show the whole menu.

After creating the menu categories, you can add them to the main menu. To support this, the **ToolStrip** class is equipped with a property named **Items** and it makes this property available to the **MenuStrip** class. The **Items** property is of type **ToolStripItemCollection**. This class implements the **IList**, the **ICollection**, and the **IEnumerable** interfaces. Therefore, to add a menu category to a **MenuStrip** object, you can call the **Add()** method of the **ToolStripItemCollection** class. This method is overloaded with various versions and we saw that one of them uses the following version:

```
Public Function Add(value As ToolStripItem) As Integer
```

You can call this version and pass it a **ToolStripItem**-type of object, such as a **ToolStripMenuItem** value. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

        Private MenuMain As MenuStrip
        Private MenuFile As ToolStripMenuItem
        Private MenuFileNew As ToolStripMenuItem
        Private MenuFileOpen As ToolStripMenuItem
        Private MenuFileExit As ToolStripMenuItem
        Private MenuEdit As ToolStripMenuItem
        Private MenuEditCopy As ToolStripMenuItem
        Private MenuEditPaste As ToolStripMenuItem
        Private MenuHelp As ToolStripMenuItem

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            MenuMain = New MenuStrip

            MenuFile = New ToolStripMenuItem("File")
            MenuFileNew = New ToolStripMenuItem("New")
            MenuFileOpen = New ToolStripMenuItem("Open")
            MenuFileExit = New ToolStripMenuItem("Exit")
            Dim MenuFileItems() As ToolStripMenuItem = _
                {MenuFileNew, MenuFileOpen, MenuFileExit}
            MenuFile.DropDownItems.AddRange(MenuFileItems)

            MenuEdit = New ToolStripMenuItem("Edit")
            MenuEditCopy = New ToolStripMenuItem("Copy")
            MenuEdit.DropDownItems.Add(MenuEditCopy)
            MenuEditPaste = CType(MenuEdit.DropDownItems.Add("Paste"), _
                ToolStripMenuItem)

            MenuHelp = New ToolStripMenuItem("Help")
            Dim mnuHelpItems() As ToolStripMenuItem = _
                { _
                    New ToolStripMenuItem("Search"), _
                    New ToolStripMenuItem("Contents"), _
                    New ToolStripMenuItem("Index"), _
                    New ToolStripMenuItem("Support Web Site"), _
                    New ToolStripMenuItem("About this application") _
                }
            MenuHelp.DropDownItems.AddRange(mnuHelpItems)

            MenuMain.Items.Add(MenuFile)
            Controls.Add(MenuMain)
        End Sub

    End Class

    Function Main() As Integer

        Dim frmStart As Starter = New Starter

        Application.Run(frmStart)

        Return 0
    End Function
```

End Module

In the same way, you can add the other items. Alternatively, you can create an array of menu categories and add them in a row. To support this, the **ToolStripItemCollection** is equipped with the **AddRange()** method that is overloaded with two versions. One of the versions uses the following syntax:

```
Public Sub AddRange(toolStripItems As ToolStripItem())
```

When calling this method, pass it an array of **ToolStripItem** types of values. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuMain As MenuStrip
    Private MenuFile As ToolStripMenuItem
    Private MenuFileNew As ToolStripMenuItem
    Private MenuFileOpen As ToolStripMenuItem
    Private MenuFileExit As ToolStripMenuItem
    Private MenuEdit As ToolStripMenuItem
    Private MenuEditCopy As ToolStripMenuItem
    Private MenuEditPaste As ToolStripMenuItem

    Private MenuView As ToolStripMenuItem

    Private MenuHelp As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        MenuMain = New MenuStrip

        MenuFile = New ToolStripMenuItem("File")
        MenuFileNew = New ToolStripMenuItem("New")
        MenuFileOpen = New ToolStripMenuItem("Open")
        MenuFileExit = New ToolStripMenuItem("Exit")
        Dim MenuFileItems() As ToolStripMenuItem = _
            {MenuFileNew, MenuFileOpen, MenuFileExit}
        MenuFile.DropDownItems.AddRange(MenuFileItems)

        MenuEdit = New ToolStripMenuItem("Edit")
        MenuEditCopy = New ToolStripMenuItem("Copy")
        MenuEdit.DropDownItems.Add(MenuEditCopy)
        MenuEditPaste = CType(MenuEdit.DropDownItems.Add("Paste"), _
            ToolStripMenuItem)

        MenuView = New ToolStripMenuItem("View")
        Dim MenuViewItems() As ToolStripMenuItem = _
            { _
                New ToolStripMenuItem("Standard Toolbar"), _
                New ToolStripMenuItem("Formatting Toolbar"), _
                New ToolStripMenuItem("Status Bar") _
            }
        MenuView.DropDownItems.AddRange(MenuViewItems)

        MenuHelp = New ToolStripMenuItem("Help")
        Dim mnuHelpItems() As ToolStripMenuItem = _
            { _
                New ToolStripMenuItem("Search"), _
                New ToolStripMenuItem("Contents"), _
                New ToolStripMenuItem("Index"), _
                New ToolStripMenuItem("Support Web Site"), _
                New ToolStripMenuItem("About this application") _
            }
        MenuHelp.DropDownItems.AddRange(mnuHelpItems)

        Dim mnuAccessories() As ToolStripMenuItem = {MenuView, MenuHelp}

        MenuMain.Items.Add(MenuFile)
        MenuMain.Items.AddRange(mnuAccessories)

        Controls.Add(MenuMain)
    End Sub
End Class
```

```

        End Sub

    End Class

    Function Main() As Integer

        Dim frmStart As Starter = New Starter

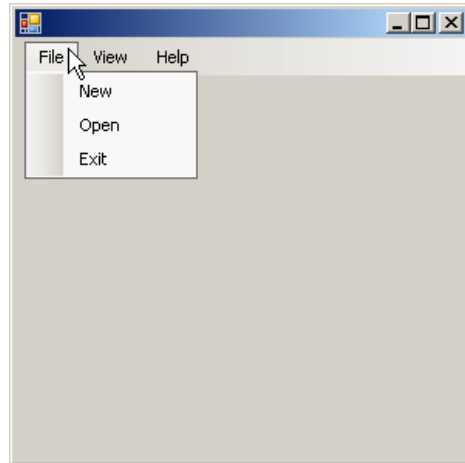
        Application.Run(frmStart)

        Return 0
    End Function


End Module

```

This would produce:



❖ Practical Learning: Creating a Menu

1. Start Microsoft [Visual Basic](#)
2. Create a new Windows Application named **AltairRealtors3**
3. In the Solution Explorer, right-click Form1.vb and click Rename
4. Type **AltairRealtors.vb** and press Enter
5. From the Menus & Toolbars section of the Toolbox, click the MenuStrip button  and click the form
6. While the menu strip is still selected, in the Properties window, click (Name), type **MenuMain** and press Enter
7. On the form, click Type Here, type **File** and press Enter
8. On the form, click File.
In the Properties window, click (Name) and type **MenuFile**
9. On the form, click File and under it, click the Type Here box
10. Type **New Property** and press Enter
11. On the form, click File and click New Property.
In the Properties window, click (Name) and type **MenuFileNewProperty**
12. On the form, click File and, under New Property, click the Type Here box
13. Type **Exit** and press Enter
14. On the form, click File and click Exit.
In the Properties window, click (Name) and type **MenuFileExit**
15. Complete the design of the form as follows:

Prop #	Property Type	Address	City	State	ZIP Code	Beds	Baths	Market Value
602138	Single Family	11604 Aldora Avenue	Baltimore	MD	21205	5	3.5	265880
749562	Townhouse	495 Parker House Terr...	Gettysburg	WV	26201	3	2.5	225500
304750	Condominium	5900 24th Street NW ...	Washington	DC	20008	1	1.0	388665
682630	Single Family	6114 Costinna Avenue	Martinsburg	WV	25401	4	3.5	325000
480750	Condominium	10710 Desprello Street...	Rockville	MD	20856	1	1.0	528445
209475	Single Family	519D Estuardo Way	Silver Spring	MD	20906	2	1.0	325995

Control	Text	Name	Other Properties	
MenuStrip				
Label	Altair Realtors - Properties Listing		Font: Times New Roman, 21.75pt, style=Bold ForeColor: MediumBlue	
ListView		lvwProperties		
Columns	(Name)	Text	TextAlign	Width
	colPropertyNumber	Prop #		50
	colPropertyType	Property Type		78
	colAddress	Address		130
	colCity	City		80
	colState	State		40
	colZIPCode	ZIP Code	Center	58
	colBedrooms	Beds	Right	40
	colBathrooms	Baths	Right	40
	colMarketValue	Market Value	Right	75

- On the main menu, click Project -> Add Windows Form...
- Set the Name to **RealEstateProperty** and click Add
- Design the form as follows:

Control	Text	Name	Other Properties
Label	Property #:		
TextBox		txtPropertyNumber	Modifiers: Public
Label	Property Type:		
ComboBox		cbxPropertyTypes	Modifiers: Public Items: Unknown Single Family Townhouse Condominium

Label	A	Address:		
TextBox	abl		txtAddress	Modifiers: Public
Label	A	City:		
TextBox	abl		txtCity	Modifiers: Public
Label	A	State:		
ComboBox	abl		cbxStates	Modifiers: Public Items: DC MD PA VA WV
Label	A	ZIP Code:		
TextBox	abl		txtZIPCode	Modifiers: Public
Label	A	Bedrooms:		
TextBox	abl	0	txtBedrooms	Modifiers: Public
Label	A	Bathrooms:		
TextBox	abl	1.0	txtBathrooms	Modifiers: Public
Label	A	Market Value:		
TextBox	abl	0.00	txtMarketValue	Modifiers: Public
Button	ab	OK	btnOK	DialogResult: OK
Button	ab	Cancel	btnCancel	DialogResult: Cancel

Form

FormBorderStyle: FixedDialog
 Text: Altair Realtors - Available Property
 StartPosition: CenterScreen
 AcceptButton: btnOK
 CancelButton: btnCancel
 MaximizeBox: False
 MinimizeBox: False
 ShowInTaskBar: False

19. Display the AltairRealtors form

Coding a Menu Item

If you create a menu as we have just done, to write code for one of the menu items, you can double-click the menu item. This would open the Click event of the menu item in the Code Editor and you can start writing the desired code for that item.

❖ Practical Learning: Writing Code For a Main Menu

1. Right-click the form and click View Code
2. In the Class Name combo box, select MenuFileNewProperty
3. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub MenuFileNewProperty_Click(ByVal sender As Object, _
                                     ByVal e As System.EventArgs) _
    Handles MenuFileNewProperty.Click

    Dim rndNumber As Random
    Dim number1 As Integer
    Dim number2 As Integer
    Dim PropertyNumber As String
    Dim dlgProperty As RealEstateProperty

    rndNumber = New Random(DateTime.Now.Millisecond)
    number1 = rndNumber.Next(100, 999)
  
```

```

number2 = rndNumber.Next(100, 999)
PropertyNumber = number1 & number2
dlgProperty = New RealEstateProperty

dlgProperty.txtPropertyNumber.Text = PropertyNumber
dlgProperty.Text = "Altair Realtors - New Property"

If dlgProperty.ShowDialog() = Windows.Forms.DialogResult.OK Then
    Dim strPropertyType As String = dlgProperty.cbxPropertyTypes.Text
    Dim strAddress As String = dlgProperty.txtAddress.Text
    Dim strCity As String = dlgProperty.txtCity.Text
    Dim strState As String = dlgProperty.cbxStates.Text
    Dim strZIPCde As String = dlgProperty.txtZIPCode.Text
    Dim strBedrooms As String = dlgProperty.txtBedrooms.Text
    Dim strBathrooms As String = dlgProperty.txtBathrooms.Text
    Dim strMarketValue As String = dlgProperty.txtMarketValue.Text

    Dim lviProperty As ListViewItem = _
        New ListViewItem(dlgProperty.txtPropertyNumber.Text)

    lviProperty.SubItems.Add(strPropertyType)
    lviProperty.SubItems.Add(strAddress)
    lviProperty.SubItems.Add(strCity)
    lviProperty.SubItems.Add(strState)
    lviProperty.SubItems.Add(strZIPCde)
    lviProperty.SubItems.Add(strBedrooms)
    lviProperty.SubItems.Add(strBathrooms)
    lviProperty.SubItems.Add(strMarketValue)
    lvwProperties.Items.Add(lviProperty)
End If
End Sub

```

- In the Class Name combo box, select MenuFileExit
- In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub MenuFileExit_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles MenuFileExit.Click

    End
End Sub

```

- Execute the application



- To close the form, click File -> Exit



[Home](#)

Copyright © 2008 FunctionX, Inc.



Applications Menus: The Contextual Menu

Contextual Menus

Introduction

In our introduction to the main menu, we saw that, to access it, the user clicks one of its categories. A contextual menu is one that appears when the user right-clicks an area of an application or form. In most applications, when the user right-clicks a title bar, the operating system is configured to display a system menu. Here is an example:

Shop for Menu

Designer Wine Accessories. In Stock! Easy Shipping Worldwide.
Lumiliving.com/Menu

Free Flash Menus

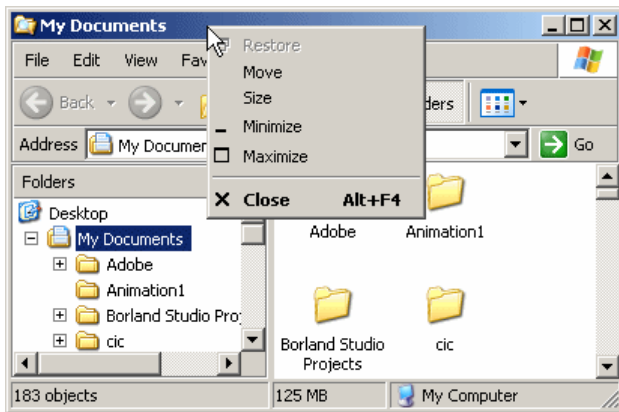
Easy to create animated menus for your website. Don't miss it!
www.FlashVortex.com

Slash Voice-Over Costs

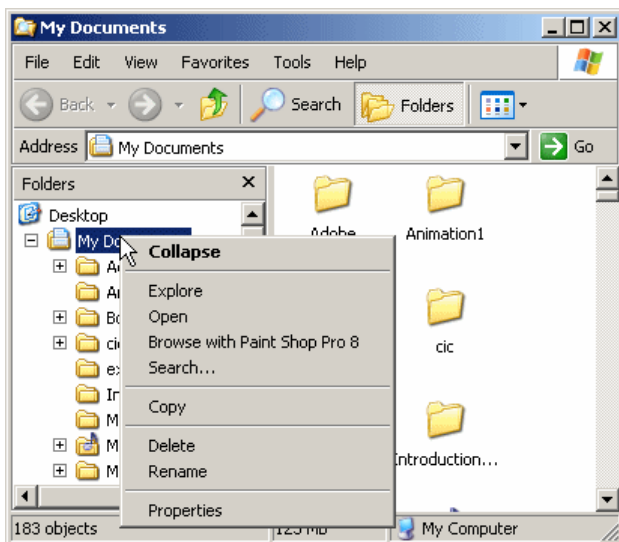
Add High-Quality, Low-Cost Voice- Overs in E-Learning Presentations.
www.speechover.com

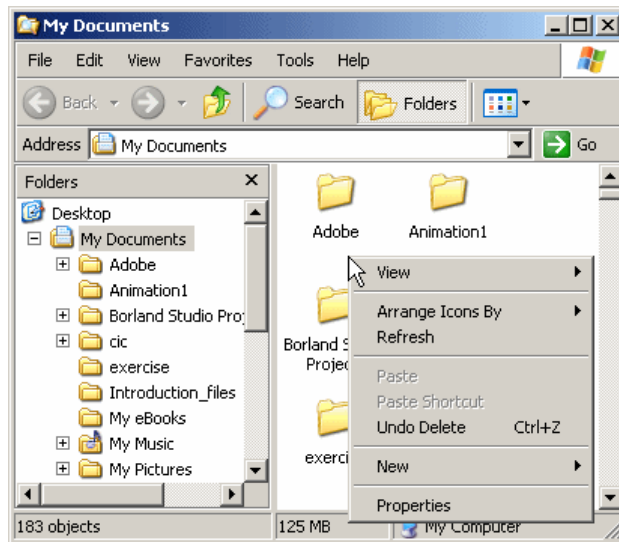


Ads by Google




A menu is considered, or qualifies as, popup if, or because, it can appear anywhere on the form as the programmer wishes. Such a menu is also referred to as context-sensitive or contextual because its appearance and behavior depend on where it displays on the form or on a particular control. The person who creates the application decides if or where the contextual menu would appear. Because this characteristic is up to the programmer, the same application can display different types of popup menus depending on where the user right-clicks. Here are examples:







The first difference between a main menu and a popup menu is that a popup menu appears as one category or one list of items and not like a group of categories of menus like a main menu. Secondly, while a main menu by default is positioned on the top section of a form, a popup menu doesn't have a specific location on the form.

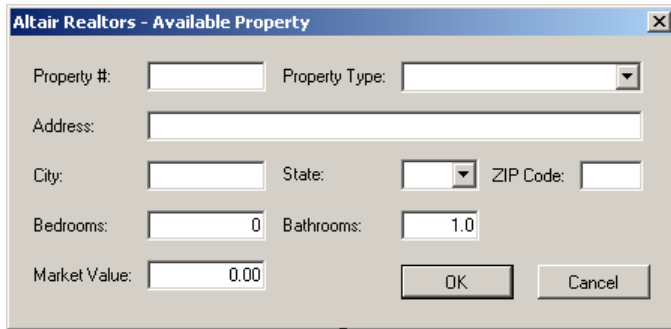
❖ Practical Learning: Creating a Menu






1. Start Microsoft Visual Basic
2. Create a new Windows Application named **AltairRealtors3b**
3. In the Solution Explorer, right-click Form1.vb and click Rename
4. Type **AltairRealtors.vb** and press Enter
5. From the Menus & Toolbars section of the Toolbox, click the MenuStrip button  and click the form
6. While the menu strip is still selected, in the Properties window, click (Name), type **MenuMain** and press Enter
7. On the form, click Type Here, type **File** and press Enter
8. On the form, click File.
In the Properties window, click (Name) and type **MenuFile**
9. On the form, click File and under it, click the Type Here box
10. Type **New Property** and press Enter
11. On the form, click File and click New Property.
In the Properties window, click (Name) and type **MenuFileNewProperty**
12. On the form, click File and, under New Property, click the Type Here box
13. Type **Exit** and press Enter
14. On the form, click File and click Exit.
In the Properties window, click (Name) and type **MenuFileExit**
15. Complete the design of the form as follows:

Prop #	Property Type	Address	City	State	ZIP Code	Beds	Baths	Market Value
602138	Single Family	11604 Aldora Avenue	Baltimore	MD	21205	5	3.5	265880
749562	Townhouse	495 Parker House Terr...	Gettysburg	WV	26201	3	2.5	225500
304750	Condominium	5900 24th Street NW ...	Washington	DC	20008	1	1.0	388665
682630	Single Family	6114 Costinna Avenue	Martinsburg	WV	25401	4	3.5	325000
480750	Condominium	10710 Desprello Street...	Rockville	MD	20856	1	1.0	528445
209475	Single Family	519D Estuardo Way	Silver Spring	MD	20906	2	1.0	325995

Control	Text	Name	Other Properties	
MenuStrip				
Label	A Altair Realtors - Properties Listing		Font: Times New Roman, 21.75pt, style=Bold ForeColor: MediumBlue	
ListView		lvwProperties		
Columns	(Name)	Text	TextAlign	Width
	colPropertyNumber	Prop #		50
	colPropertyType	Property Type		78
	colAddress	Address		130
	colCity	City		80
	colState	State		40
	colZIPCode	ZIP Code	Center	58
	colBedrooms	Beds	Right	40
	colBathrooms	Baths	Right	40
colMarketValue	Market Value	Right	75	

16. On the main menu, click Project -> Add Windows Form...
17. Set the Name to **RealEstateProperty** and click Add
18. Design the form as follows:



Control	Text	Name	Other Properties
Label	A Property #:		
TextBox		txtPropertyNumber	Modifiers: Public
Label	A Property Type:		
ComboBox		cbxPropertyTypes	Modifiers: Public Items: Unknown Single Family Townhouse Condominium
Label	A Address:		
TextBox		txtAddress	Modifiers: Public
Label	A City:		
TextBox		txtCity	Modifiers: Public
Label	A State:		
ComboBox		cbxStates	Modifiers: Public Items: DC MD PA VA WV
Label	A ZIP Code:		
TextBox		txtZIPCode	Modifiers: Public

	abl			
Label	A	Bedrooms:		
TextBox	abl	0	txtBedrooms	Modifiers: Public
Label	A	Bathrooms:		
TextBox	abl	1.0	txtBathrooms	Modifiers: Public
Label	A	Market Value:		
TextBox	abl	0.00	txtMarketValue	Modifiers: Public
Button	ab	OK	btnOK	DialogResult: OK
Button	ab	Cancel	btnCancel	DialogResult: Cancel

Form

FormBorderStyle: FixedDialog
 Text: Altair Realtors - Available Property
 StartPosition: CenterScreen
 AcceptButton: btnOK
 CancelButton: btnCancel
 MaximizeBox: False
 MinimizeBox: False
 ShowInTaskBar: False

19. Display the AltairRealtors form
20. Right-click the form and click View Code
21. In the Class Name combo box, select MenuFileNewProperty
22. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub MenuFileNewProperty_Click(ByVal sender As Object, _
                                     ByVal e As System.EventArgs) _
    Handles MenuFileNewProperty.Click

    Dim rndNumber As Random
    Dim number1 As Integer
    Dim number2 As Integer
    Dim PropertyNumber As String
    Dim dlgProperty As RealEstateProperty

    rndNumber = New Random(DateTime.Now.Millisecond)
    number1 = rndNumber.Next(100, 999)
    number2 = rndNumber.Next(100, 999)
    PropertyNumber = number1 & number2
    dlgProperty = New RealEstateProperty

    dlgProperty.txtPropertyNumber.Text = PropertyNumber
    dlgProperty.Text = "Altair Realtors - New Property"

    If dlgProperty.ShowDialog() = Windows.Forms.DialogResult.OK Then
        Dim strPropertyType As String = dlgProperty.cbxPropertyTypes.Text
        Dim strAddress As String = dlgProperty.txtAddress.Text
        Dim strCity As String = dlgProperty.txtCity.Text
        Dim strState As String = dlgProperty.cbxStates.Text
        Dim strZIPCde As String = dlgProperty.txtZIPCode.Text
        Dim strBedrooms As String = dlgProperty.txtBedrooms.Text
        Dim strBathrooms As String = dlgProperty.txtBathrooms.Text
        Dim strMarketValue As String = dlgProperty.txtMarketValue.Text

        Dim lviProperty As ListViewItem = _
            New ListViewItem(dlgProperty.txtPropertyNumber.Text)

        lviProperty.SubItems.Add(strPropertyType)
        lviProperty.SubItems.Add(strAddress)
        lviProperty.SubItems.Add(strCity)
        lviProperty.SubItems.Add(strState)
        lviProperty.SubItems.Add(strZIPCde)
        lviProperty.SubItems.Add(strBedrooms)
        lviProperty.SubItems.Add(strBathrooms)
        lviProperty.SubItems.Add(strMarketValue)
        lvwProperties.Items.Add(lviProperty)
    End If
End Sub
  
```

23. In the Class Name combo box, select MenuFileExit

24. In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub MenuFileExit_Click(ByVal sender As Object, _
                              ByVal e As System.EventArgs) _
    Handles MenuFileExit.Click
End Sub
```

25. Execute the application




26. To close the form, click File -> Exit

Creating a Contextual Menu

To support the creation and management of contextual menus, the .NET Framework provides the **ContextMenuStrip** class. This class is derived from **ToolStripDropDownMenu**, which itself is based on the **ToolStripDropDown** class. The **ToolStripDropDown** class is derived from **ToolStrip**.

To visually create a contextual menu, in the Menus & Toolbars section of the Toolbox, click the

ContextMenuStrip button  and click the form. Once you have a **ContextMenuStrip** object, you can create its menu items. To do this, as mentioned for the **MenuStrip**, you can click the first Type Here line, type a string, press Enter, and continue creating the other menu items in the same way.

Unlike a main menu, a popup menu provides a single list of items. If you want different popup menus for your form, you have two options. You can create various popup menus or programmatically change your single popup menu in response to something or some action on your form.

To programmatically create a contextual menu, start by declaring a handle to **ContextMenuStrip**. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter
    Inherits Form

    Private Contextual As ContextMenuStrip

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()

        Contextual = New ContextMenuStrip

    End Sub

End Class

Function Main() As Integer

    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function
```


End Module

To assist you with each item of a contextual menu, **ToolStrip**, the ancestor to the **ContextMenuStrip** class, is equipped with a property named **Items**. This property is of type **ToolStripItemCollection**, which is a collection-based class. The **ToolStripItemCollection** class implements the **IList**, the **ICollection**, and the **IEnumerable** interfaces.

To create one or more menu items, you can use the various techniques we reviewed for the main menu. Here are examples:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter
    Inherits Form

    Private MenuEditCut As ToolStripMenuItem
    Private Contextual As ContextMenuStrip

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()

        Contextual = New ContextMenuStrip
        MenuEditCut = New ToolStripMenuItem("Cut")

        Dim MenuEdit() As ToolStripMenuItem = _
        { _
            New ToolStripMenuItem("Copy"), _
            New ToolStripMenuItem("Paste") _
        }

    End Sub

End Class

Function Main() As Integer

    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function
```

End Module

After creating a menu item, to add it to the contextual menu, you can call the **ToolStripItemCollection.Add()** method. To add an array of items, you can call the create **ToolStripItemCollection.AddRange()** method. Here are examples:

```
Public Sub InitializeComponent()

    Contextual = New ContextMenuStrip
    MenuEditCut = New ToolStripMenuItem("Cut")

    Dim MenuEdit() As ToolStripMenuItem = _
    { _
        New ToolStripMenuItem("Copy"), _
        New ToolStripMenuItem("Paste") _
    }

    Contextual.Items.Add(MenuEditCut)
    Contextual.Items.AddRange(MenuEdit)

End Sub
```

❖ Practical Learning: Introducing Contextual Menus


1. From the Menus & Toolbars section of the Toolbox, click ContextMenuStrip and click the form
2. While the context menu strip is still selected, in the Properties window click (Name) and type **MenuWithProperties**
3. Then click Items and click its ellipsis button

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

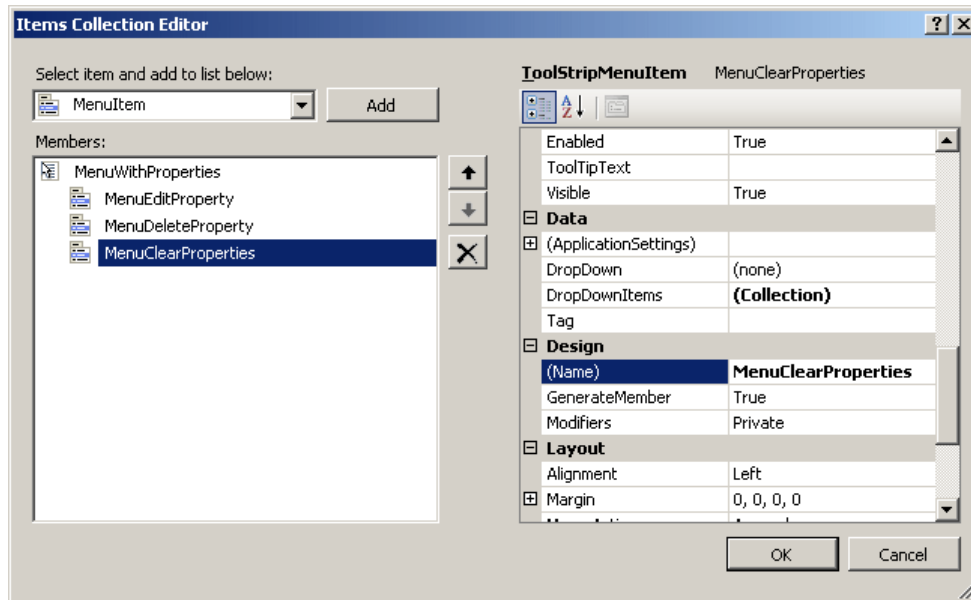
VIEW PLANS >



MANASHOSTING

www.manashosting.com
Ads by Google

4. Under the Select Item And Add To List combo box, make sure MenuItem is selected and click Add
5. On the right side, click Text and type **Edit**
6. Click (Name) and type **MenuEditProperty**
7. On the left side, click Add and, on the right side, change the properties as follows:
Text: **Delete**
(Name): **MenuDeleteProperty**
8. On the left side, click Add and, on the right side, change the properties as follows:
Text: **Clear**
(Name): **MenuClearProperties**



9. Click OK
10. From the Menus & Toolbars section of the Toolbox, click ContextMenuStrip and click the form
11. While the context menu strip is still selected, in the Properties window click (Name) and type **MenuNoProperty**
12. On the form, under ContextMenuStrip, click Type Here
13. Type **New Property** and press Enter
14. On the form, under ContextMenuStrip, click New Property
15. In the Properties window, click (Name), type **MenuNewProperty** and press Enter

Using a Contextual Menu

By default, a newly created contextual menu is attached neither to the form nor to any control on it. In order to display a contextual menu, you must assign its name to the control. To support this, **Control**, the ancestor to all visual controls of the .NET Framework, is equipped, and provides to its children, a property named **ContextMenuStrip**, which is of type **ContextMenuStrip**.

To visually assign a contextual menu to a control during design, click the control. In the Properties window, click the ContextMenuStrip field, then click the arrow of its combo box, and select the menu. If you had created more than one contextual menu, the combo box would show all of them and you can choose the one you want to use as default.

To programmatically specify the contextual menu of a control, assign a **ContextMenuStrip** object to its **ContextMenuStrip** property. Here is an example:

```
Public Sub InitializeComponent()

    Contextual = New ContextMenuStrip
    MenuEditCut = New ToolStripMenuItem("Cut")

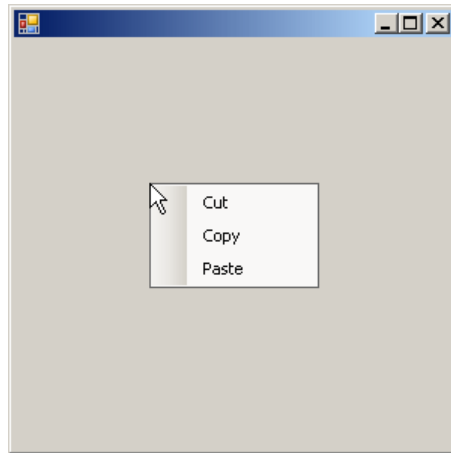
    Dim MenuEdit() As ToolStripMenuItem = _
    {
        New ToolStripMenuItem("Copy"), _
        New ToolStripMenuItem("Paste") _
    }

    Contextual.Items.Add(MenuEditCut)
    Contextual.Items.AddRange(MenuEdit)
End Sub
```

```
ContextMenuStrip = Contextual
```

```
End Sub
```

After assigning a **ContextMenuStrip** object to a control, when you right-click (actually when the user right-clicks) the control, the contextual menu would display. The above code would produce:



❖ Practical Learning: Creating a Context Menu

1. Right-click the form and click View Code
2. Just under the Public Class AltairRealtors line, declare a ListViewItem variable named **itmSelected**

```
Public Class AltairRealtors
    Private itmSelected As ListViewItem
    . . . No Change
```

3. In the Class Name combo box, select (AltairRealtors Events)
4. In the Method Name combo box, select Load and implement the event as follows:

```
Private Sub AltairRealtors_Load(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
    Handles Me.Load
    itmSelected = New ListViewItem()
    lvwProperties.ContextMenuStrip = MenuNoProperty
End Sub
```

5. In the Class Name combo box, select lvwProperties
6. In the Method Name combo box, select MouseDown and implement the event as follows:

```
Private Sub lvwProperties_MouseDown(ByVal sender As Object, _
                                    ByVal e As System.Windows.Forms.MouseEventArgs) _
    Handles lvwProperties.MouseDown
    If e.Button = Windows.Forms.MouseButtons.Right Then
        If lvwProperties.SelectedItems.Count > 0 Then
            itmSelected = lvwProperties.SelectedItems(0)
        Else
            itmSelected = Nothing
        End If
    End If
End Sub
```

7. Save the file

Coding Contextual Menus

In your application, you can create as many contextual menus as you want. If you have different controls, each can have its own contextual menu or many can share a contextual menu. Also, you can use different contextual menus for a control and decide what menu to display when/why.

There is nothing particularly specific with writing code for a popup menu item. You approach it exactly as if you were dealing with a menu item of a main menu. You can write code for an item of a popup menu independent of any other item of a main menu. If you want an item of a popup menu to respond to the same request as an item of a main menu, you can write code for

one of the menu items (either the item on the main menu or the item on the popup menu) and simply call its Click event in the event of the other menu item.

❖ Practical Learning: Using Various Contextual Menus

1. In the Class Name combo box, select `lvwProperties`
2. In the Method Name combo box, select `ItemSelectionChanged` and implement the event as follows:

```
Private Sub lvwProperties_MouseDown(ByVal sender As Object, _
                                   ByVal e As System.Windows.Forms.MouseEventArgs) _
                                   Handles lvwProperties.MouseDown
    If e.Button = Windows.Forms.MouseButtons.Right Then
        If lvwProperties.SelectedItems.Count > 0 Then
            itmSelected = lvwProperties.SelectedItems(0)
        Else
            itmSelected = Nothing
        End If
    End If
End Sub
```

3. In the Class Name combo box, select `MenuEditProperties`
4. In the Method Name combo box, select `Click` and implement the event as follows:

```
Private Sub MenuEditProperty_Click(ByVal sender As Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles MenuEditProperty.Click
    ' Prepare to open the AvailableProperties dialog box
    Dim dlgProperty As RealEstateProperty = New RealEstateProperty

    ' Make sure an item, and only one, is selected
    If (lvwProperties.SelectedItems.Count = 0) Or _
        (lvwProperties.SelectedItems.Count > 1) Then
        Exit Sub
    End If

    ' Identify the item that is currently selected
    Dim lviCurrent As ListViewItem = lvwProperties.SelectedItems(0)

    ' Display the ItemDetails dialog box with the item number
    dlgProperty.txtPropertyNumber.Text = lviCurrent.Text
    dlgProperty.cbxPropertyTypes.Text = lviCurrent.SubItems(1).Text
    dlgProperty.txtAddress.Text = lviCurrent.SubItems(2).Text
    dlgProperty.txtCity.Text = lviCurrent.SubItems(3).Text
    dlgProperty.cbxStates.Text = lviCurrent.SubItems(4).Text
    dlgProperty.txtZIPCode.Text = lviCurrent.SubItems(5).Text
    dlgProperty.txtBedrooms.Text = lviCurrent.SubItems(6).Text
    dlgProperty.txtBathrooms.Text = lviCurrent.SubItems(7).Text
    dlgProperty.txtMarketValue.Text = lviCurrent.SubItems(8).Text

    If dlgProperty.ShowDialog() = Windows.Forms.DialogResult.OK Then
        lvwProperties.SelectedItems(0).Text = _
            dlgProperty.txtPropertyNumber.Text
        lvwProperties.SelectedItems(0).SubItems(1).Text = _
            dlgProperty.cbxPropertyTypes.Text
        lvwProperties.SelectedItems(0).SubItems(2).Text = _
            dlgProperty.txtAddress.Text
        lvwProperties.SelectedItems(0).SubItems(3).Text = _
            dlgProperty.txtCity.Text
        lvwProperties.SelectedItems(0).SubItems(4).Text = _
            dlgProperty.cbxStates.Text
        lvwProperties.SelectedItems(0).SubItems(5).Text = _
            dlgProperty.txtZIPCode.Text
        lvwProperties.SelectedItems(0).SubItems(6).Text = _
            dlgProperty.txtBedrooms.Text
        lvwProperties.SelectedItems(0).SubItems(7).Text = _
            dlgProperty.txtBathrooms.Text
        lvwProperties.SelectedItems(0).SubItems(8).Text = _
            dlgProperty.txtMarketValue.Text
    End If
End Sub
```

5. In the Class Name combo box, select `MenuDeleteProperty`
6. In the Method Name combo box, select `Click` and implement the event as follows:

```
Private Sub MenuDeleteProperty_Click(ByVal sender As Object, _
                                      ByVal e As System.EventArgs) _
                                      Handles MenuDeleteProperty.Click
    If lvwProperties.SelectedItems.Count = 0 Then Exit Sub
    Dim Answer As MsgBoxResult

    Answer = MsgBox("Are you sure you want " & _
```

```

        "to delete that property?", _
        MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
        "Delete Property")

    If Answer = MsgBoxResult.Yes Then
        lvwProperties.SelectedItems(0).Remove()
    End If
End Sub

```

7. In the Class Name combo box, select MenuClearProperty
8. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub MenuClearProperties_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles MenuClearProperties.Click
    Dim Answer As MsgBoxResult
    Answer = MsgBox("Are you sure you want " & _
        "to delete all properties?", _
        MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
        "Remove all Properties")

    If Answer = MsgBoxResult.Yes Then
        lvwProperties.Items.Clear()
    End If
End Sub

```

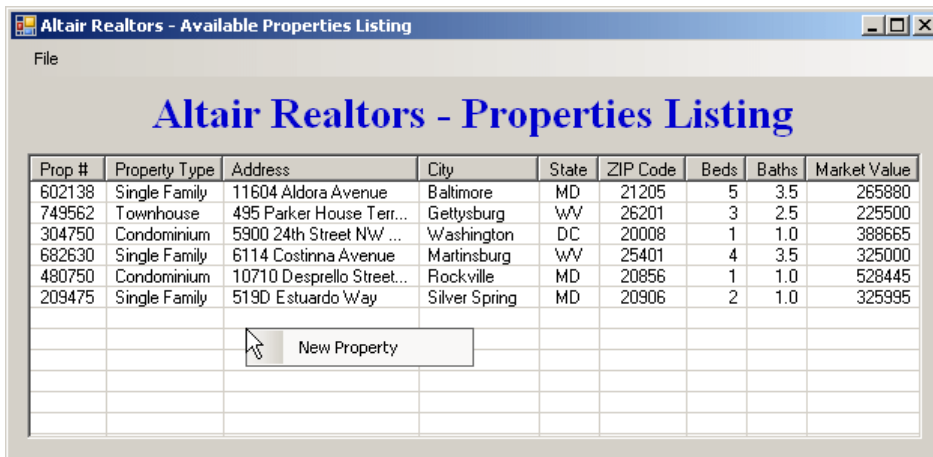
9. In the Class Name combo box, select MenuNewProperty
10. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub MenuNewProperty_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles MenuNewProperty.Click
    MenuFileNewProperty_Click(sender, e)
End Sub

```

11. Execute the application and test it
12. Right-click an empty line of the list view to see the contextual menu and click New Property



Altair Realtors - New Property

Property #: 364-790 Property Type: Townhouse

Address: 10422 Carrito Avenue

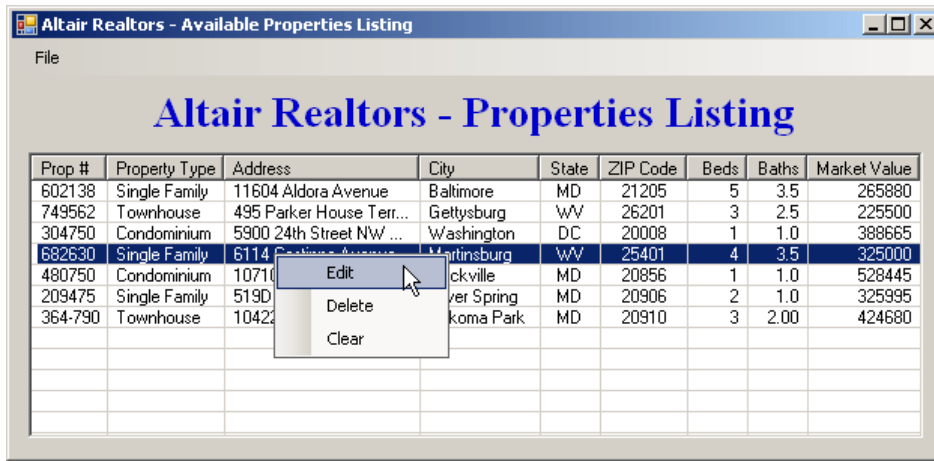
City: Takoma Park State: MD ZIP Code: 20910

Bedrooms: 3 Bathrooms: 2.00

Market Value: 424680

OK Cancel

13. Right-click the list view and click Edit



Property #: 682630 Property Type: Single Family

Address: 6114 Costin Avenue

City: Martinsburg State: WV ZIP Code: 25401

Bedrooms: 5 Bathrooms: 2.0

Market Value: 412680

OK Cancel

14. Right-click a row on the form and click Delete
15. Accept to delete the property
16. Close the form and return to your programming environment

CD Front End Get this versatile CD authoring software: autorun CD presentations, brochures, catalogues... Professional, MSWindows.

www.CDFrontEnd.com

Feedback - Arts by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.



Characteristics of Menu Items

Introduction

In the previous lesson, we saw how to create a menu. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter
    Inherits Form

    Private mnuMain As MenuStrip

    Private mnuFile As ToolStripMenuItem
    Private mnuFileNew As ToolStripMenuItem

    Dim components As System.ComponentModel.IContainer

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()

        mnuMain = New MenuStrip
        Controls.Add(mnuMain)

        mnuFile = New ToolStripMenuItem("File")
        mnuFileNew = New ToolStripMenuItem("New")

        mnuFile.DropDownItems.Add(mnuFileNew)
        mnuMain.Items.Add(mnuFile)

    End Sub

End Class

Function Main() As Integer

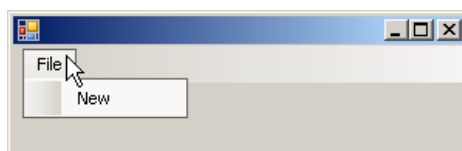
    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function
```

End Module

This would produce:



After creating a menu (main menu and contextual menu), there are various actions you can perform to improve it and there are many ways you can enhance the user's experience with your application. Menus provide various features such as access keys and shortcuts. There are also other things you can do such as grouping menus. Although some of these actions are not required to make an application useful, they can be necessary to make it more professional.

❖ Practical Learning: Introducing Menu Appearance

1. Start a new Windows Forms Application named **SolasPropertyRental2**

www.manashosting.com

Ads by Google

2. In the Solution Explorer, right-click Form1.vb and click Rename
3. Type **Central.vb** and press Enter twice (to display the form)
4. Change the properties of the form as follows:
Text: **Solas Property Rental**
StartPosition: CenterScreen
5. On the main menu, click Project -> Add Class...
6. Set the Name to **RentalProperty** and click Add
7. Change the file as follows:

```
Public Class RentalProperty
    Private code As String
    Private type As String
    Private beds As Integer
    Private baths As Single
    Private rent As Double
    Private status As String

    Public Property PropertyCode() As String
        Get
            Return code
        End Get
        Set(ByVal value As String)
            code = value
        End Set
    End Property

    Public Property PropertyType() As String
        Get
            Return type
        End Get
        Set(ByVal value As String)
            type = value
        End Set
    End Property

    Public Property Bedrooms() As Integer
        Get
            Return beds
        End Get
        Set(ByVal value As Integer)
            beds = value
        End Set
    End Property

    Public Property Bathrooms() As Single
        Get
            Return baths
        End Get
        Set(ByVal value As Single)
            baths = value
        End Set
    End Property

    Public Property MonthlyRent() As Double
        Get
            Return rent
        End Get
        Set(ByVal value As Double)
            rent = value
        End Set
    End Property

    Public Property OccupancyStatus() As String
        Get
            Return status
        End Get
        Set(ByVal value As String)
            status = value
        End Set
    End Property

End Class
```

8. In the Solution Explorer, right-click Central.vb and click View Code
9. Just above the first line, type **Imports System.Collections** and, in the class, declare an **ArrayList** variable named **lstRentalProperties**


```
Imports System.Collections
```




```
Public Class Central
    Private lstRentalProperties As ArrayList
End Class
```

10. In the Class Name combo box, select (Central Events)
11. In the Method Name combo box, select Load and initialize the variable as follows:

```
Private Sub Central_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles Me.Load
    lstRentalProperties = New ArrayList
End Sub
```

12. Re-display the Central form
13. In the Menus & Toolbars section of the Toolbox, click the MenuStrip button  and click the form
14. While the menu strip is still selected, in the Properties window, click (Name) and type **mnuMain**
15. In the Common Controls section of the Toolbox, click ListView and click the form
16. While the picture box is still selected, in the Properties window, change its characteristics as follows:
 Dock: **Fill**
 FullRowSelect: **True**
 GridLines: **True**
 (Name): **lvwRentalProperties**
 View: **Details**
 HeaderStyle: **Nonclickable**
17. Still in the Properties window, click Columns and click its ellipsis button
18. Create the columns as follows:

(Name)	Text	TextAlign	Width
colPropertyCode	Prop Code		
colPropertyType	Property Type	Center	80
colBedrooms	Bedrooms	Right	
colBathrooms	Bathrooms	Right	62
colMonthlyRent	Monthly Rent	Right	75
colStatus	Status		

19. Click OK
20. In the Menus & Toolbars section of the Toolbox, click the ContextMenuStrip button  and click the form
21. While the menu strip is still selected, in the Properties window, click (Name) and type **cmsProperties**
22. On the form, click the list view
23. In the Properties window, click ContextMenuStrip and select cmsProperties

Access Keys

You may notice that some [menu items](#) have a letter underlined. Using this letter allows the user to access the menu using a keyboard. For example, if the letter F is underline in a File menu as in File, the user can access the File menu by pressing the Alt, then the F keys. To create this functionality, choose a letter on the menu item and precede it with the & character. For example, &File would produce File. You can apply the same principle if you are programmatically creating the menu. Here are two examples:

```
Public Sub InitializeComponent()

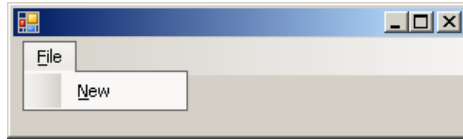
    mnuMain = New MenuStrip
    Controls.Add(mnuMain)

    mnuFile = New ToolStripMenuItem("&File")
    mnuFileNew = New ToolStripMenuItem("&New")

    mnuFile.DropDownItems.Add(mnuFileNew)
    mnuMain.Items.Add(mnuFile)

End Sub
```

After creating the menu, to use it, the user can press Alt or F10:



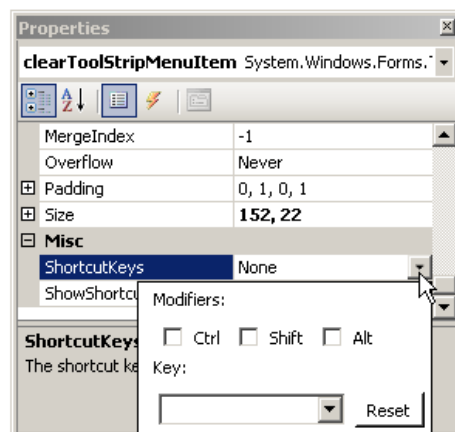
❖ Practical Learning: Using Access Keys

1. Under the form, click mnuMain
2. In the Properties window, click Items and click its ellipsis button
3. In the Items Collection Editor, make sure MenuItem is selected in the Select Item And Add To List Below combo box and click Add
4. While toolStripMenuItem1 is selected in the Members combo box, in the right list, change the following characteristics:
Text: **&File**
(Name): **mnuFile**
5. Still in the right list, click DropDownItems and click its ellipsis button
6. In the Items Collection Editor, make sure MenuItem is selected in the Select Item And Add To List Below combo box and click Add
7. While toolStripMenuItem1 is selected in the Members combo box, in the right list, change the following characteristics:
Text: **&New Property**
(Name): **mnuFileNewProperty**
8. In the Items Collection Editor (mnuFile.DropDownItems), click OK
9. In the Items Collection Editor, click OK

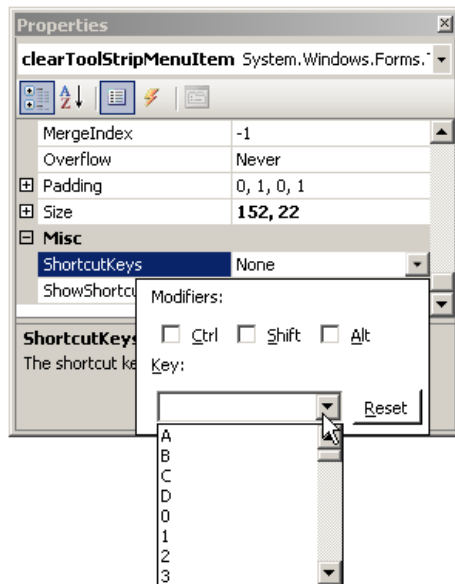
Shortcuts

A shortcut is a key or a combination of keys that the user can press to perform an action that can also be performed using a menu item. When creating a menu, to specify a shortcut, use the **ShortcutKeys** property.

To visually specify a shortcut, in the menu [designer](#), click the menu item. In the Properties window, click ShortcutKeys and click the arrow of the field, a window would come up:



To specify just a letter for the shortcut, you can click the arrow of the combo box on the left side of the Reset button. A list would come up from which you can select the desired letter:



You are probably more familiar with shortcuts made of combinations of keys, such as Ctrl + N, Alt + F6, or Ctrl + Alt + Delete. To visually create such a shortcut, click the check box(es) and select the desired letter.

If you have used applications like [Microsoft Word](#) or Adobe Photoshop, you may know that they don't show all of their shortcuts on the menu. If you want to hide a shortcut, after specifying it, in the Properties window, set the **ShowShortcutKeys** property to **False**.

To programmatically specify a shortcut, assign a key or a combination of keys to the **ShortcutKeys** property of the **ToolStripMenuItem** class. The **ShortcutKeys** property is of type **Keys**, which is an enumeration of the various keys of a keyboard recognized by [Microsoft Windows](#). Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms
```

Module [Exercise](#)

```
Public Class Starter
    Inherits Form

    Private mnuMain As MenuStrip
    Private mnuFile As ToolStripMenuItem
    Private mnuFileNew As ToolStripMenuItem
    Private mnuFileExit As ToolStripMenuItem
    Private mnuFormat As ToolStripMenuItem
    Private mnuFormatFont As ToolStripMenuItem

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()

        mnuMain = New MenuStrip
        Controls.Add(mnuMain)

        mnuFile = New ToolStripMenuItem("&File")
        mnuFileNew = New ToolStripMenuItem("&New")

        mnuFileExit = New ToolStripMenuItem("E&xit")

        mnuFormat = New ToolStripMenuItem("For&mat")
        mnuFormatFont = New ToolStripMenuItem("Fo&nt")

        mnuFormatFont.ShortcutKeys = Keys.F4

        mnuFile.DropDownItems.Add(mnuFileNew)
        mnuFile.DropDownItems.Add(mnuFileExit)
        mnuMain.Items.Add(mnuFile)

        mnuFormat.DropDownItems.Add(mnuFormatFont)
        mnuMain.Items.Add(mnuFormat)
```

```

End Sub

End Class

Function Main() As Integer

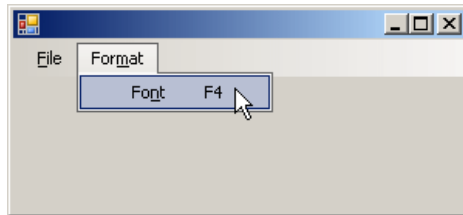
    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function

End Module
    
```

This would produce:



To create a shortcut that is a combination of keys, use the bit manipulation operator OR represented by |. Here is an example:

```

Public Sub InitializeComponent()

    mnuMain = New MenuStrip
    Controls.Add(mnuMain)

    mnuFile = New ToolStripMenuItem("&File")
    mnuFileNew = New ToolStripMenuItem("&New")

    mnuFileNew.ShortcutKeys = Keys.Control Or Keys.N

    mnuFileExit = New ToolStripMenuItem("E&xit")

    mnuFormat = New ToolStripMenuItem("For&mat")
    mnuFormatFont = New ToolStripMenuItem("Fo&nt")

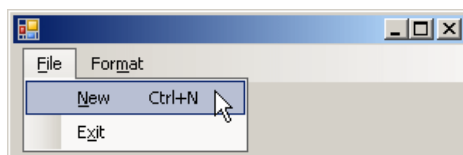
    mnuFormatFont.ShortcutKeys = Keys.F4

    mnuFile.DropDownItems.Add(mnuFileNew)
    mnuFile.DropDownItems.Add(mnuFileExit)
    mnuMain.Items.Add(mnuFile)

    mnuFormat.DropDownItems.Add(mnuFormatFont)
    mnuMain.Items.Add(mnuFormat)

End Sub
    
```

This would produce:



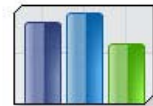
Normally, when you have associated a shortcut with a menu item, when the user displays the menu, the shortcut would appear. In some applications, you may want to hide the shortcut. To support this, the **ToolStripMenuItem** class is equipped with the Boolean **ShowShortcutKeys** property. The default value of this property is true. If you want to hide the shortcut, you can set this property to false.

❖ Practical Learning: Creating Shortcuts

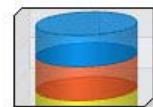
1. Under the form, click mnuMain
2. In the Properties window, click Items and click its ellipsis button
3. In the Members list of the Items Collection Editor, click mnuFile
4. On the right side, click DropDownItems and click its ellipsis button



FREE DEVELOPER VERSION
 DOWNLOAD NOW



.netCHARTING
 helps you look
 your best.

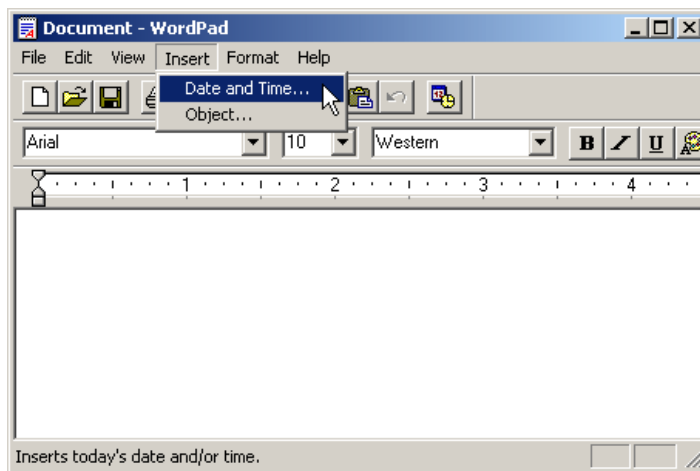


www.dotnetcharting.com
 Ads by Google

5. In the Members list, click mnuFileNewProperty
6. In the right list, click **ShortcutKeys** and click the arrow of its combo box
7. In the window that appears, click the Ctrl check box
8. Click the arrow of the combo box next to Reset, scroll down and select **N**
9. In the Items Collection Editor (mnuTools.DropDownItems), click OK
10. In the Items Collection Editor, click OK

Three Periods

When a user has clicked a menu item, an action is supposed to occur. In some cases, an intermediary action is necessary before performing or completing the action. To indicate that an intermediary action is needed for the action related to the menu item, [Microsoft](#) standards suggest that the menu's text be followed by three periods. For example, in WordPad, if you want to display the date or the time or both on a document, you must open a dialog box that would present various options for you to choose how the date/time should be displayed. To indicate that you will perform a primary action before displaying the value, the menu that leads to it shows three periods:



In this case, when you click the menu item, a dialog box would come up for you to select the desired value.

There is no programmatic [relationship](#) between the application and the menu item that displays three periods. It is only a suggestion to show them. Therefore, when creating a menu item, if you know that an intermediary action will be used to perform or complete the actual action, add three periods on the right side of its text. Here is an example:

```
Imports System.Drawing
```

```
Imports System.Windows.Forms
```

```
Module Exercise
```

```
    Public Class Starter
```

```
        Inherits Form
```

```
        Private mnuMain As MenuStrip
```

```
        Private mnuSelect As ToolStripMenuItem
```

```
        Private mnuSelectColor As ToolStripMenuItem
```

```
        Dim components As System.ComponentModel.Container
```

```
        Public Sub New()
```

```
            InitializeComponent()
```

```
        End Sub
```

```
        Public Sub InitializeComponent()
```

```
            mnuMain = New MenuStrip
```

```
            Controls.Add(mnuMain)
```

```
            mnuSelect = New ToolStripMenuItem("&Select")
```

```
            mnuSelectColor = New ToolStripMenuItem("<u>Background Color...</u>")
```

```
            mnuSelect.DropDownItems.Add(mnuSelectColor)
```

```
            mnuMain.Items.Add(mnuSelect)
```

```

    End Sub

End Class

Function Main() As Integer

    Dim frmStart As Starter = New Starter

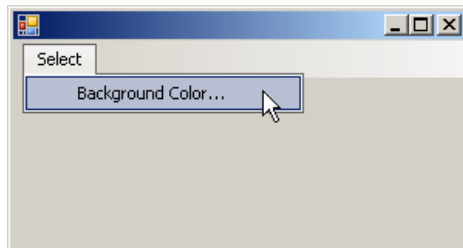
    Application.Run(frmStart)

    Return 0
End Function

End Module

```

This would produce:



Because the three periods indicate to the user that an intermediary action will be performed, when implementing the code for the menu item, make sure you provide that intermediary action. Here is an example:

```

Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

        Private mnuMain As MenuStrip
        Private mnuSelect As ToolStripMenuItem
        Friend WithEvents mnuSelectColor As ToolStripMenuItem

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()

            mnuMain = New MenuStrip
            Controls.Add(mnuMain)

            mnuSelect = New ToolStripMenuItem("&Select")

            mnuSelectColor = New ToolStripMenuItem("Background Color...")

            mnuSelect.DropDownItems.Add(mnuSelectColor)
            mnuMain.Items.Add(mnuSelect)

        End Sub

        Private Sub SelectBackgroundColor(ByVal sender As Object, _
                                         ByVal e As EventArgs) _
            Handles mnuSelectColor.Click

            Dim dlgColor As ColorDialog = New ColorDialog

            If dlgColor.ShowDialog() = Windows.Forms.DialogResult.OK Then
                BackColor = dlgColor.Color
            End If
        End Sub
    End Class

Function Main() As Integer

    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

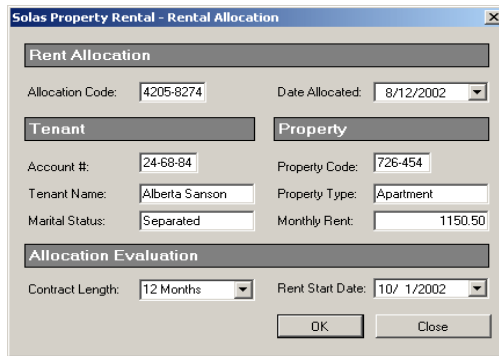
```

Return 0
End Function

End Module

❖ Practical Learning: Creating an Intermediary Action

1. Under the form, click cmsProperties
2. In the Properties window, click Items and click its ellipsis button
3. In the Select Items And Add To List Below combo box, make sure MenuItem is selected and click Add
4. On the right side, click Text and type **New Property...**
5. Click (Name) and type **mnuProperty**
6. Click Shortcut and click the arrow of its combo box
7. Click the Ctrl check box and click the arrow of the combo box to select N
8. In the Items Collection Editor, click OK
9. On the main menu, click Project -> Add Windows Form
10. Set the Name to **PropertyEditor** and click Add
11. Design the form as follows:



Control	Text	Name	Other Properties
Label	Property Code:		
TextBox		txtPropertyCode	
Button	OK	btnOK	DialogResult: OK
Label		Property Type:	
ComboBox	Unknown	cbxPropertyTypes	Items: Unknown Apartment Townhouse Single Family
Button	Cancel	btnCancel	DialogResult: Cancel
Label	Bedrooms:		
TextBox	0	txtBedrooms	
Label	Bathrooms:		
TextBox	0.00	txtBathrooms	
Label	Monthly Rent:		
TextBox	0.00	txtMonthlyRent	
Label	Occupancy Status:		
ComboBox	Unknown	cbxStatus	Unknown Available Occupied Needs Repair

Form FormBorderStyle: FixedDialog
 Text: Solas Property [Rental - Property](#) Editor
 StartPosition: CenterScreen
 AcceptButton: btnOK
 CancelButton: btnCancel
 MaximizeBox: False
 MinimizeBox: False

ShowInTaskbar: False

12. In the Solution Explorer, right-click Central.vb and click View Code
13. In the Class Name combo box, select mnuProperty
14. In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub mnuProperty_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs) _
    Handles mnuProperty.Click
    Dim RandomCode As Random
    Dim Editor As PropertyEditor
    Dim SampleProperty As RentalProperty
    Dim strCode1 As String, strCode2 As String

    RandomCode = New Random
    strCode1 = CStr(RandomCode.Next(100, 999))
    strCode2 = CStr(RandomCode.Next(100, 999))

    Editor = New PropertyEditor
    Editor.txtPropertyCode.Text = strCode1 & "-" & strCode2

    If Editor.ShowDialog() = Windows.Forms.DialogResult.OK Then
        SampleProperty = New RentalProperty
        SampleProperty.PropertyCode = Editor.txtPropertyCode.Text
        SampleProperty.PropertyType = Editor.cbxPropertyTypes.Text
        SampleProperty.Bedrooms = CInt(Editor.txtBedrooms.Text)
        SampleProperty.Bathrooms = CSng(Editor.txtBathrooms.Text)
        SampleProperty.MonthlyRent = CDb1(Editor.txtMonthlyRent.Text)
        SampleProperty.OccupancyStatus= Editor.cbxStatus.Text
        lstRentalProperties.Add(SampleProperty)
    End If

    lvwRentalProperties.Items.Clear()

    If lstRentalProperties.Count > 0 Then
        For Each prop As RentalProperty In lstRentalProperties
            Dim itmProperty As ListViewItem = _
                New ListViewItem(prop.PropertyCode)
            itmProperty.SubItems.Add(prop.PropertyType)
            itmProperty.SubItems.Add(CStr(prop.Bedrooms))
            itmProperty.SubItems.Add(FormatNumber(prop.Bathrooms))
            itmProperty.SubItems.Add(FormatNumber(prop.MonthlyRent))
            itmProperty.SubItems.Add(prop.OccupancyStatus)
            lvwRentalProperties.Items.Add(itmProperty)
        Next
    End If
End Sub
```

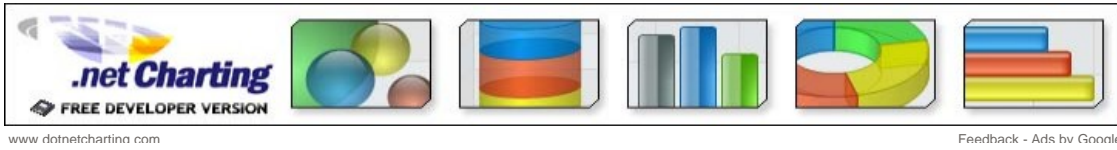
15. Return to the Central form
16. On the form, click File and click New Property
17. In the Properties window, edit its **Text** property to display **&New Property...**
18. On the form, click File and double-click New Property
19. Implement the event as follows:

```
Private Sub mnuFileNewProperty_Click(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
    Handles mnuFileNewProperty.Click
    mnuProperty_Click(sender, e)
End Sub
```

20. Execute the application and try creating the following properties (let the [computer](#) generate the properties codes):

Property Types	Bedrooms	Bathrooms	Monthly Rent	Status
Apartment	1	1	925	Occupied
Apartment	2	1	1150.50	Available
Single Family	5	3.5	2250.85	Occupied
Townhouse	3	2.5	1750	Occupied
Townhouse	4	2.5	1920.50	Available
Single Family	4	2.5	2140.50	Needs Repair
Apartment	3	2	1250.25	Available
Townhouse	3	1.5	1650.50	Occupied

21. Close the form and return to your [programming environment](#)



.net Charting
FREE DEVELOPER VERSION

www.dotnetcharting.com

Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

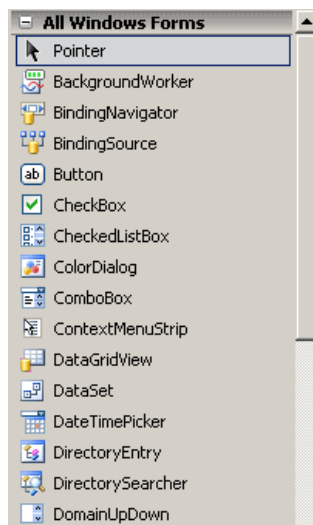
[Next](#)



Studio Windows: The Toolbox

Introduction

A Windows control is a graphical object that allows the user to interact with the computer. The controls are as varied as the needs and goals are. Because there are so many controls for various purposes, their insertion to an application and their configuration are left to the computer programmer. The Toolbox is the accessory that provides most of the controls used in an application:



[Free Datagrid for WPF](#)

100% stylable and templatable, with rich in-place editing & more

xceed.com/Grid_WPF_Intro.html



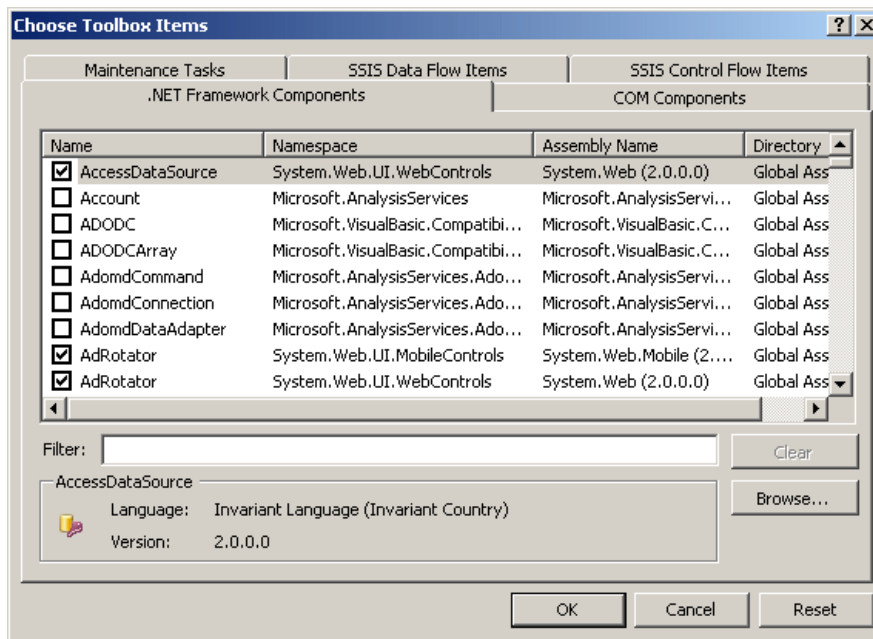
Ads by Google

By default, the Toolbox is positioned on the left side of the IDE. To change that position, you can drag its title bar away and dock it to another side of the IDE. The Toolbox also uses a default width to show the items on it. If the width is too narrow or too wide, you can change it. To do this, position the mouse to its right border and drag left or right.

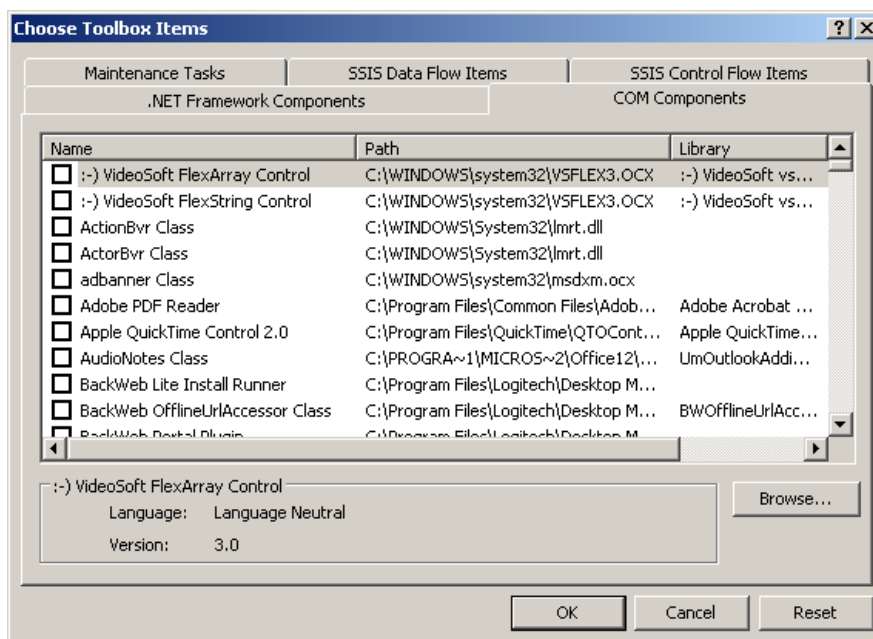
The Toolbox and Additional Controls

When Microsoft Visual Studio is set up, it installs in the Toolbox the most regularly used controls. If you are working in an environment that creates only a particular group of applications and there are controls you hardly use, if you want, you can remove them from the list. To remove a control, right-click it and click Delete.

Besides the objects in the Common Controls section, other controls are left out but are still available. Some of the left out controls were created with the .NET Framework but are not installed by default because they are judged hardly used. To add one or more of these left out controls, right-click anywhere in the Toolbox and click Choose Items... In the Choose Toolbox Items dialog box, click the .NET Framework Components tab, then click the check box of the desired control before clicking OK:

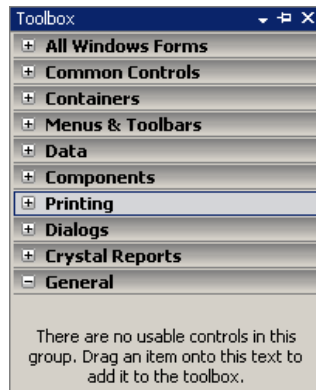


In addition to custom .NET controls, some other objects called ActiveX controls were used in previous versions of Visual Basic or Visual Studio and are available. To take care of compatibility issues, most previous ActiveX controls were reconfigured and adapted to be used in a .NET application. To add some of these left out controls, right-click anywhere in the Toolbox and click Choose Items... In the Choose Toolbox Items dialog box, click the COM Components tab, select the desired control before clicking OK

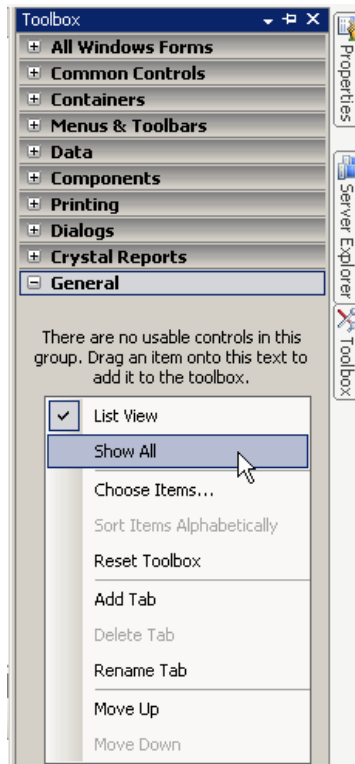


The Sections of the Toolbox

When you start a Windows Application, it provides various controls on the Toolbox so you can choose which ones are appropriate for your particular application. Controls can be set by categories based on their function or role. A container is a control whose main purpose is to host other controls. To design it, you pick up objects from the Toolbox and drop them where desired. The Toolbox organizes its items in categories and each category is represented by a button:



If the available list of categories is not enough, you can add a new section of your choice. By default, Visual Studio hides some categories because they are judged hardly used. To display these additional sections, you can right-click anywhere in the Toolbox and click Show All:



If you still want an additional tab not included in the list, you can add one (or more). To do that, right-click anywhere in the Toolbox and click Add Tab. You would be prompted to provide a name. After typing the new name, press Enter.

The Layout of a Category

To use an object of a particular category, you can first click its button. After selecting a category, it displays its items. In each category, a particular button called Pointer is selected by default. This simply indicates that no item in the group is currently selected.

By default, the items in each category are organized as horizontal wide buttons:

MANASHOSTING



MANASHOSTING

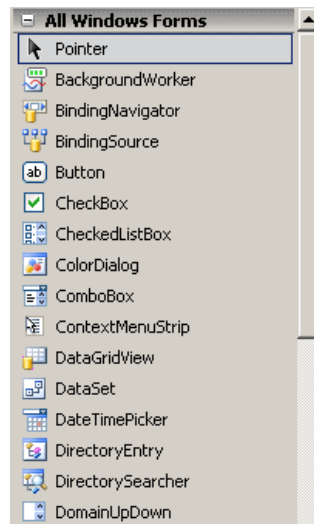
Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

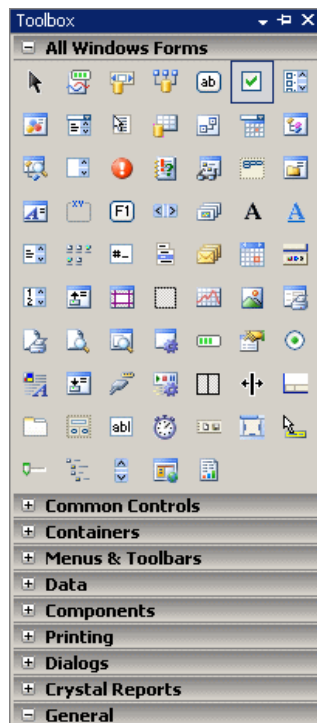
Only Rs.2000/-
Per Year

VIEW PLANS >

www.manashosting.com
Ads by Googale

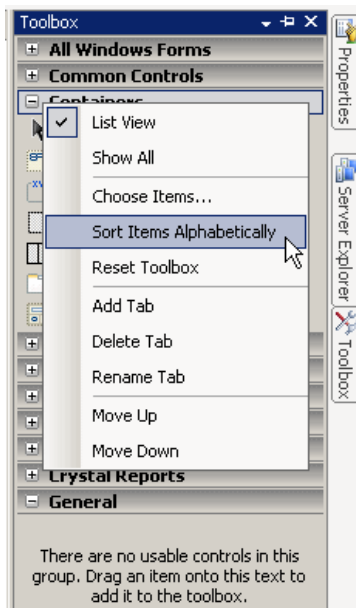


Alternatively, you can list the items of a category as buttons of a list view. To do that, you can right-click anywhere in the category and click List View to remove its check box:



Arrangement of Items in the Toolbox

When Microsoft Visual Studio is installed, it adds the buttons in a somewhat random order. In the beginning, this can make it difficult to find a particular control when you need it. If you find it more convenient, you can arrange the list of controls in any order of your choice. You have two main options. To change the position of an item in the list, right-click it and click either Move Up or Move Down. Alternatively, you can arrange the items in alphabetic order. To do that, right-click anywhere in the Windows Forms section and click Sort Items Alphabetically:



Once you have rearranged items alphabetically, the Toolbox forgets the previous arrangement and you cannot restore it. Alternatively, you can right-click the button of a control and click either Move Up or Move Down.

Unlimited Webspace and Unlimited Bandwidth		FREE	Only
⇒ 100 Email Ids of 10 GB	⇒ Full Control Panel	Domain Registration	Rs.2000/-
⇒ All Database FREE	⇒ 200 SEO Tools/Softwares/Weblinks		
⇒ 5000 Templates / Website Builder			

www.manashosting.com

Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



Studio Object: The Properties Window

Introduction

A property is a piece of information that characterizes or describes a control. It could be related to its location or size. It could be its color, its identification, or any visual aspect that gives it meaning. The properties of an object can be changed either at design time or at run time. You can also manipulate these characteristics both at design and at run times. This means that you can set some properties at design time and some others at run time.

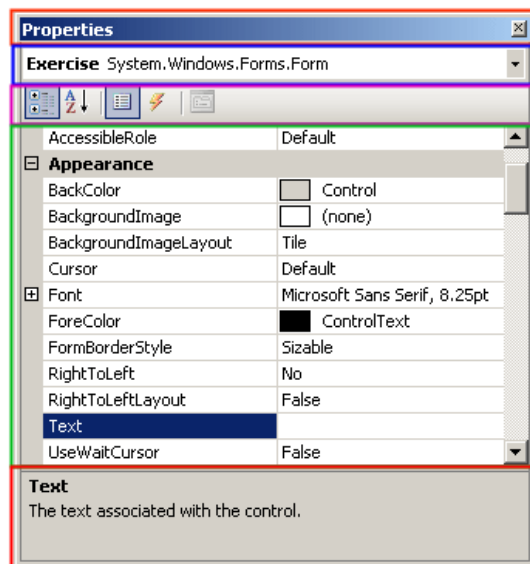
To manipulate the properties of a control at design time, first select it on the form. While a control is selected, use the Properties window to manipulate the properties of the control at design time. To access the Properties window if it is not visible:




- On the main menu, you can click View -> Properties Window
- On the form, you can right-click anything (either the form itself or any control positioned on it) and click Properties
- The shortcut to display the Properties window is F4

Description

The Properties window uses the behaviors we reviewed in Lesson 1 about auto-hiding, docking, floating or tabbing the tools that accompany Microsoft Visual Studio 2005. This means that you can position it on one side of the screen or to have it float on the screen as you wish.

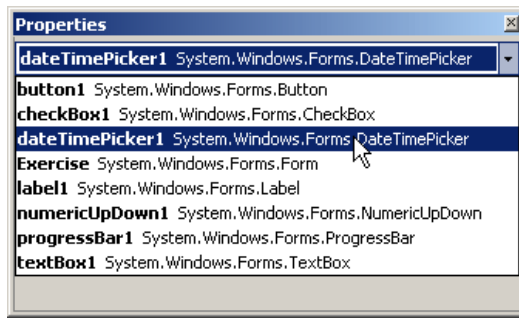
The Properties window is divided in 5 sections:



The Properties window starts on top with a title bar, which displays the string Properties. If the window is docked somewhere, it displays the Window Position , the Auto-Hide , and the Close  buttons on its right side. If the window is floating, it would display only the Close button.

Under the title bar, the Properties window displays a combo box. The content of the combo box is the name of the form plus the names of the controls currently on the form. Besides the technique we reviewed earlier to select a control, you can click the arrow of the combo box and

select a control from the list:



Under the combo box, the Properties displays a toolbar with 4 buttons.

Under the toolbar, the Properties window displays the list of properties of the selected control (s). On the right side, the list is equipped with a vertical scroll bar. The items in the Properties window display in a list set when installing Microsoft Visual Studio. In the beginning, you may be regularly lost when looking for a particular property because the list is not arranged in a strict order of rules. You can rearrange the list. For example, you can cause the items to display in alphabetic order. To do this, on the toolbar of the Properties window, click the Alphabetic button . To restore the list, you can click the categorized button .

Two lists share the main area of the Properties window. When the list of properties is displaying, the Properties button is clicked . The second is the list of events. Therefore, to show the events, you can click the Events button . If the events section is displaying, to show the list of properties, you can click the Properties button .

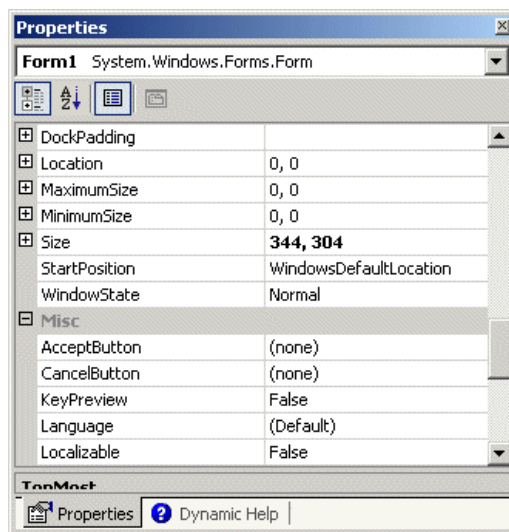
Under the list of properties, there is a long bar that displays some messages. The area is actually a help section that displays a short description of the property that is selected in the main area of the Properties window.

Accessing the Properties of One or More Controls

Based on a previous description,

- If the Properties window is already displaying, to access the properties of the form or of a control, simply click it
- If the Properties window is not displaying, to access the characteristics of an object, right-click either the form or a control on the form and click Properties
- If the Properties window is not available, to access the characteristics, click either the form or a control on the form and, on the main menu, click View -> Properties

When a control is selected, the Properties window displays only its characteristics:



You can also change some characteristics of various controls at the same time. To do this, first select the controls on the form and access the Properties window:

Unlimited Pack

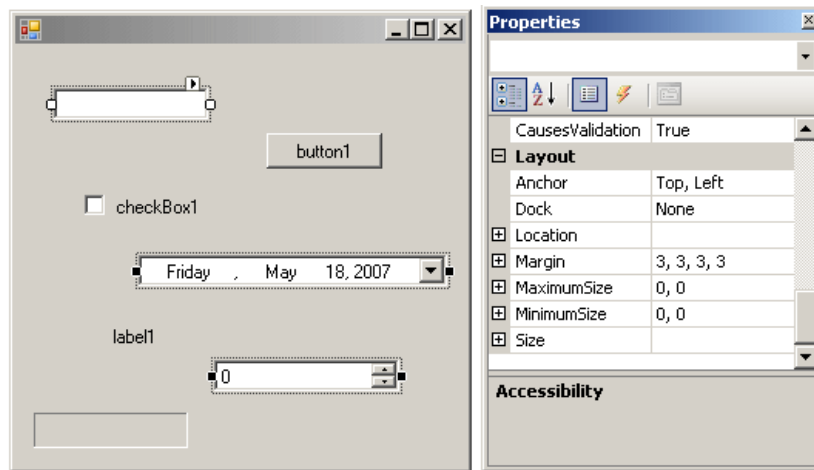
- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >

MANASHOSTING

www.manashosting.com
Ads by Goale



When various controls have been selected:

- The Properties window displays only the characteristics that are common to the selected controls
- The combo box on top of the Properties window is empty
- Some fields of the Properties appear empty because the various controls that are selected have different values for those properties

❖ Practical Learning: Introducing the Properties Window

1. To create a new application, on the main menu, click File -> New Project...
2. In the Templates list, click Windows Application
3. Set the Name to **Exercise4** and click OK

Unlimited Webspace and Unlimited Bandwidth		FREE	Only Rs. 2000/-
⇒ 100 Email Ids of 10 GB	⇒ All Database FREE	⇒ Full Control Panel	
⇒ 5000 Templates / Website Builder	⇒ 200 SEO Tools/Softwares/Weblinks		

www.manashosting.com Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Visual Basic Functions: The Message Box

Introduction

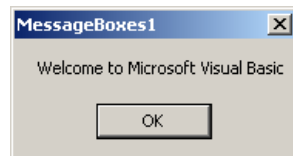
A message box is a special dialog box used to display a piece of information to the user. As opposed to a regular form, the user cannot type anything in the dialog box. To support message boxes, the [Visual Basic](#) language provides a function named **MsgBox**. To support message boxes, the [.NET Framework](#) provides a class named.

To display a simple message box, you can use the MsgBox() function with the following formula:

```
MsgBox(Message)
```

Inside the parentheses, pass a string. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MsgBox("Welcome to Microsoft Visual Basic")
End Sub
```



If the message is made of different sections, you can concatenate them using the & operator. You can also first declare a String variable, initialize it, and pass it to the function.

To create a message box using the .NET Framework, you can call the Show() method of the MessageBox [class](#) using the following formula:

```
MessageBox.Show(Message)
```

As done for the MsgBox() function, pass a string to the method. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MessageBox.Show("Welcome to Microsoft Visual Basic")
End Sub
```

In our lessons, we will mostly use the **MsgBox()** function, not because it is better than the **MessageBox** class. It is simply a preference; but it is also because these lessons are for Microsoft Visual Basic, so we give preference to its own (rich) library.

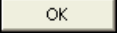
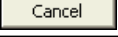

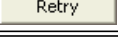

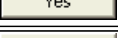
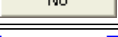
The Return Value of a Message Box

Besides displaying a message, a message box can be used to let the user make a decision by clicking a button and, depending on the button the user would have clicked, the message box would return a value. To be able to return a value, the MsgBox() function is declared as follows:

```
Public Shared Function MsgBox ( _
    Prompt As Object, _
    <OptionalAttribute> Optional Buttons As MsgBoxStyle = MsgBoxStyle.OkOnly, _
    <OptionalAttribute> Optional Title As Object = Nothing _
) As MsgBoxResult
```



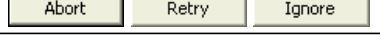
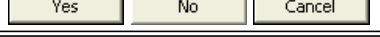


The value returned by a message box corresponds to a button the user would have clicked (on

the message box). The return value of the MsgBox() function is based on the **MsgBoxResult** enumeration. The buttons and the returned values are as follows:

If the User Clicks	Button Caption	Integral Value
	OK	1
	Cancel	2
	Abort	3
	Retry	4
	Ignore	5
	Yes	6
	No	7

The Buttons of a Message Box

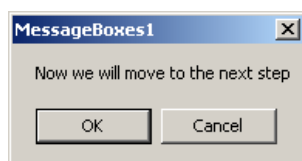
If you create a simple message box by providing only the message, it would appear with only one button labeled OK. If you want the user to be able to make a decision and communicate it to you, provide a second argument. The second argument must be based on the **MsgBoxStyle** enumeration. When it comes to buttons, some members of this enumeration are:

To Display	MsgBoxStyle	Integral Value
	OKOnly	0
	OKCancel	1
	AbortRetryIgnore	2
	YesNoCancel	3
	YesNo	4
	RetryCancel	5

To use any of these combinations of buttons, call the **MessageBoxStyle** enumeration and access the desired combination. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MsgBox("Now we will move to the next step", MsgBoxStyle.OkCancel)
End Sub
```

This would produce:



The Caption of a Message Box

If you create a simple message box by providing only the message, the dialog box would appear with the name of the project in the title. To allow you to specify a caption of your choice, provide a second string as the third argument to the **MsgBox()** function. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MsgBox("Now we will move to the next step", _
        MsgBoxStyle.OkCancel, "Lessons Objectives")
End Sub
```

This would produce:

Unlimited Pack

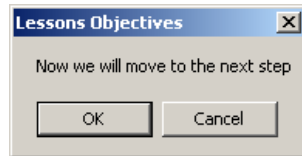
- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google



The Icon of a Message Box

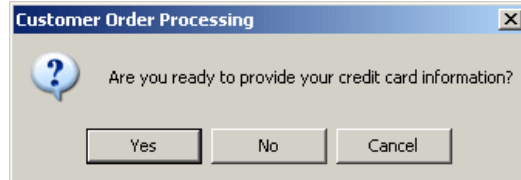
To enhance the appearance of a message box, you can display an icon on it. To support [icons](#), the **MsgBoxStyle** enumeration provides the following additional members:

To Display		MsgBoxStyle	Integral Value
		Critical	16
		Question	32
		Exclamation	48
		Information	64

To apply one of these buttons, combine its style with that of the button, using the OR operator. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MsgBox("Are you ready to provide your credit card information?", _
        MsgBoxStyle.YesNoCancel Or MsgBoxStyle.Question, _
        "Customer Order Processing")
End Sub
```

This would produce:



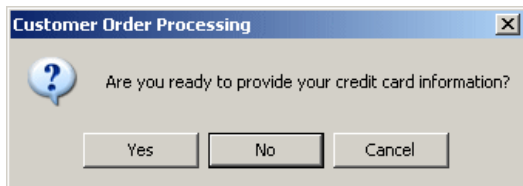
The Default Button of a Message Box

When a message box is configured to display more than one button, the [operating system](#) is set to decide which button is the default. The default button has a thick border that sets it apart from the other button(s). If the user presses Enter, the message box would behave as if the user had clicked the default button. If the message box has more than one button, you can decide what button would be the default. To support the default button, the **MsgBoxStyle** enumeration provides the following additional options:

MsgBoxStyle	Integral Value	If the message box contains more than one button, the default button would be
DefaultButton1	0	the first
DefaultButton2	256	the second
DefaultButton3	512	the third

Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    MsgBox("Are you ready to provide your credit card information?", _
        MsgBoxStyle.YesNoCancel Or _
        MsgBoxStyle.Question Or _
        MsgBoxStyle.DefaultButton2, _
        "Customer Order Processing")
End Sub
```



A better paying job balances everything!

 I.T. 11,400 jobs Search Now	 H.R. 10,956 jobs Search Now	 Finance 3,588 jobs Search Now	 B.P.O. 28,920 jobs Search Now	 Sales 10,956 jobs Search Now	Register Now >>> TIMESJOBS.COM <i>Because you are worth more</i>
--	--	--	--	---	---

[Home](#)

Copyright © 2008 FunctionX, Inc.

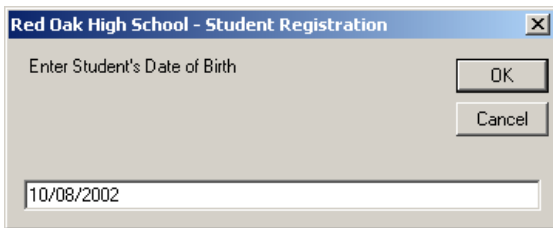
[Home](#)



Visual Basic Functions: The Input Box

Introduction

An input box is a specially designed dialog box that allows the programmer to request a value from the user and use that value as necessary. An input box displays a title, a message to indicate the requested value, a text box for the user, and two buttons: OK and Cancel. Here is an example:



When an input box displays, it presents a request to the user who can then provide a value. After using the input box, the user can change his or her mind and press Esc or click Cancel. If the user provided a value and want to acknowledge it, he or she can click OK or press Enter. This would transfer the contents of the text box to the application that displayed the input box.

Creating an Input Box

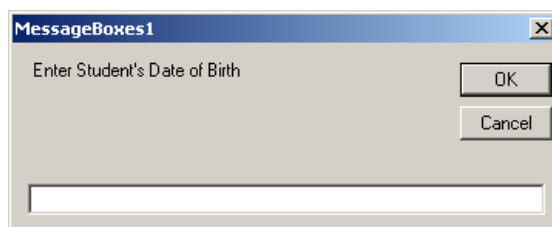
To support input boxes, the Visual Basic library provides a function named **InputBox**. Its syntax is:

```
Public Function InputBox( _
    ByVal Prompt As String, _
    Optional ByVal Title As String = "", _
    Optional ByVal DefaultResponse As String = "", _
    Optional ByVal Xpos As Integer = -1, _
    Optional ByVal YPos As Integer = -1 _
) As String
```

The only required argument of this function is the message that prompts the user. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    InputBox("Enter Student's Date of Birth")
End Sub
```

This would produce



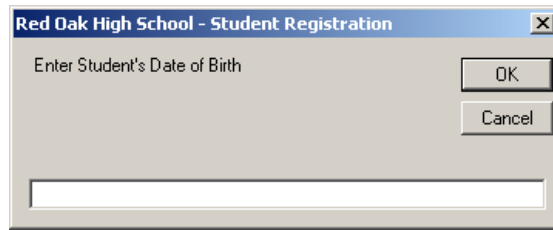
When calling the **InputBox()** function, if you pass only the first argument, the input box would display the name of the application in the title bar. If you want, you can specify your own caption through the *Title* argument. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    InputBox("Enter Student's Date of Birth", _
```

```
"Red Oak High School - Student Registration")
```

```
End Sub
```

This would produce

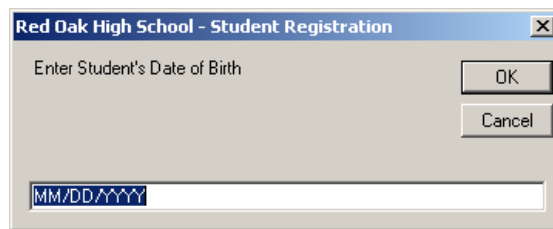


When reading the message on the Input box, the user is asked to enter a piece of information. The type of information the user is supposed to provide depends on you, the programmer. Therefore, there are two important things you should always do. First you should let the user know the type of information requested. Is it a number (what type of number)? Is it a string (such as the name of a country or a customer's name)? Is it the location of a file (such as C:\Program Files\Homework)? Are you expecting a Yes/No True/False type of answer (if so how should the user provide it)? Is it a date (if it is a date, what format is the user supposed to enter)? These questions indicate that you should state a clear request to the user.

To assist the user with the type of value you are expecting, you can give an example or a type. To support this, the **InputBox()** function is equipped with the third argument as a string. When passing it, you can provide a sample value that the user would follow. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    InputBox("Enter Student's Date of Birth", _
            "Red Oak High School - Student Registration", _
            "MM/DD/YYYY")
End Sub
```

This would produce:



The last two arguments, *XPos* and *YPos*, allow you to specify the default position of the input box when it comes up the first time.

After typing a value, the user would click one of the buttons: OK or Cancel. If the user clicks OK, you can retrieve the value the user would have typed. It is also your responsibility to find out whether the user typed a valid value or not. Because the **InputBox()** function returns a string, it has no mechanism of validating the user's entry. Therefore, if necessary, you must convert the return value of the input box when the user clicks OK. Here is an example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
    Handles btnMessage.Click
    Dim strDOB As String

    strDOB = InputBox("Enter Student's Date of Birth", _
                    "Red Oak High School - Student Registration", _
                    "MM/DD/YYYY")

    If IsDate(strDOB) Then
        MsgBox("The student was born on " & CDate(strDOB).ToString("D"), _
            MsgBoxStyle.OkOnly Or MsgBoxStyle.Information, _
            "Red Oak High School - Student Registration")
    Else
        MsgBox("You provided an invalid value", _
            MsgBoxStyle.OkOnly Or MsgBoxStyle.Information, _
            "Red Oak High School - Student Registration")
    End If
End Sub
```

Ads by Google



[Scrollable PictureBox OCX](#)

Scroll Pictures and Child Controls For VB 6 developers
Bennet-Tec.Com

[Visual Basic Code Library](#)

Open Source Code Snippet Library. Free Community for Developers.
www.daniweb.com/code

[Free Computer eBooks](#)

10,000+ Online Computer Books. all are free!
2020ok.com

[Downloadable Macro Books](#)

Get over 1200 Excel visual basic macro examples with explanations
www.add-ins.com/macro_examples

[VB to C# Converter](#)

Most Accurate VB to C# Converter. Free Demo, Tech Support & Upgrades.
www.tangiblesoftwaresolutions.com

Unlimited Webspace and Unlimited Bandwidth		FREE	Only Rs.2000/-
⇒ 100 Email Ids of 10 GB	⇒ All Database FREE	Domain Registration	
⇒ 5000 Templates / Website Builder	⇒ Full Control Panel	⇒ 200 SEO Tools/Softwares/Weblinks	

www.manashosting.com Feedback - Ads by Google

[Home](#) [Copyright © 2008 FunctionX, Inc.](#) [Home](#)



Error Handling

Introduction

Apparently no matter how careful and meticulous you are, there will be problems with your code or your [application](#). Some problems will come from you. Some problems will be caused by users. And some problems will be caused by neither you nor your users. This means that there are things you can fix. There are things you can avoid as much as possible. And there are things beyond your control. Still, as much as you can, try anticipating any type of problem you imagine may occur when a user is using your application, and take action as much as possible to avoid bad situations.



Error Categories

As mentioned above, there are three main types of problems that you will deal with, directly or indirectly:

1. **Syntax:** A syntax error comes from your mistyping a word or forming a bad expression in your code. It could be that you misspelled a keyword such as `ByVal` instead of `ByVal`. It could also be a bad expression such as `524+ + 62.55`. It could be a "grammar" error such as providing the name of a variable before its data type when declaring a variable (quite common for those who regularly transition from different languages (C/C++, Pascal, C#, [Java](#)))

When you use [Microsoft Visual Studio](#) to write your code, it would point out the errors while you are writing your code, giving up ample time to fix them. When you compile your application, the [compiler](#) can let you know about other syntax errors. For this reason, syntax errors are almost the easiest to fix. Most of the time, the compiler would point out where the problem is so you can fix it

2. **Run-Time:** After all syntax errors have been fixed, the program may be ready for the user. There are different types of problems that a user may face when interacting with your program. For example, imagine that, in your code, you indicate that a picture would be loaded and displayed to the user but you forget to ship the picture or the directory of the picture indicated in your code becomes different when a user opens your application. In this case, when you compiled and executed the application in your machine, everything was fine; but when you distribute the application and your user tries to use it, it does not work. This is a type of run-time error
3. **Logic:** These are errors that do not fit in any of the above categories. They could be caused by the user misusing your application, a problem with the [computer](#) on which the application is running while the same application is [working](#) fine in another computer. Because logic errors can be vague, they can also be difficult (even, to the extreme, impossible) to fix

One of the best qualities of an effective [programmer](#) is to anticipate as many problems as possible and to deal with them in the early stages. Some problems can be easy to fix. With some others, you will simply need to build more experience to know how to fix them. Unfortunately, it will not be unusual to have users asking you to fix your application when a problem may not come from it.

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >

MANASHOSTING

www.manashosting.com
Ads by Google



The banner features the Intel Core 2 Duo logo on the left with the text "Do More". In the center is a black Dell Inspiron 1525 laptop. To the right of the laptop is a list of features under the heading "Comes with": "Designer Patterns", "HDMI Port", and "Exciting Colors". Further right is the slogan "YOURS IS Stylish" in a mix of blue and black fonts, with a small Dell logo and "YOURS IS HERE" to its right.

[Home](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)

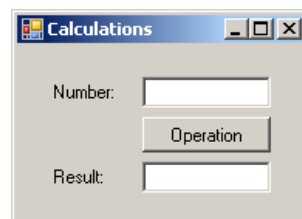


Classic Error Handling

Introduction

From its early stages, Microsoft Visual Basic has always made it a priority to deal with errors. Most or early errors occur in your code. The Visual Studio IDE can help you detect syntax errors and fix them. In fact, a feature almost unique to the Visual Basic IDE, which is not available in Visual C++ and some versions of Visual C#, is that its Code Editor detects problems immediately as soon as they appear in your code. In fact, in previous versions of Visual Basic and in VBA (Microsoft Access), a message box would display, prompting you to fix the problem. This has always made Visual Basic one of the friendliest programming environments around. When you think everything is fine, compile your code. If there is a syntax error that the IDE did not signal or that you ignored when writing your code, the compiler will let you know. If there is no syntax error, the compilation will be over and the executable will be ready. You can then execute the application to see the result. If the user is not asked to provide value(s), you are less likely to get a run-time error.

A run-time error is one that occurs when using your application. Consider the following application:



```
Imports System.Drawing
Imports System.Windows.Forms
```

Module Exercise

```
Public Class Starter
    Inherits Form

    Private lblNumber As Label
    Private txtNumber As TextBox
    Friend WithEvents btnCalculate As Button
    Private lblResult As Label
    Private txtResult As TextBox

    Dim components As System.ComponentModel.Container

    Public Sub New()
        InitializeComponent()
    End Sub

    Public Sub InitializeComponent()
        Text = "Calculations"

        lblNumber = New Label
        lblNumber.Location = New Point(17, 23)
        lblNumber.Text = "Number:"
        lblNumber.AutoSize = True

        txtNumber = New TextBox
```

```

txtNumber.Location = New Point(78, 20)
txtNumber.Size = New Size(83, 20)

btnCalculate = New Button
btnCalculate.Location = New Point(78, 45)
btnCalculate.Text = "Calculate"
btnCalculate.Size = New Size(83, 23)

lblResult = New Label
lblResult.Location = New Point(17, 75)
lblResult.Text = "Result:"
lblResult.AutoSize = True

txtResult = New TextBox
txtResult.Location = New Point(76, 72)
txtResult.Size = New Size(83, 20)

Controls.Add(lblNumber)
Controls.Add(txtNumber)
Controls.Add(btnCalculate)
Controls.Add(lblResult)
Controls.Add(txtResult)

End Sub

Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
                            Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

    Result = Number * 24
    txtResult.Text = CStr(Result)
End Sub
End Class

Function Main() As Integer

    Dim frmStart As Starter = New Starter

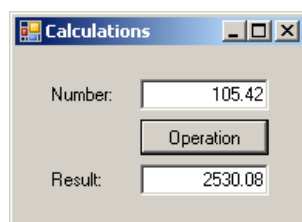
    Application.Run(frmStart)

    Return 0
End Function

End Module

```

Here is an example of executing it:



The first aspect you should take into consider is to imagine what could cause a problem. If you think there is such a possibility, start by creating a label that could be used to transfer code if a problem occurs. Here is an example:

```

Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
                            Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

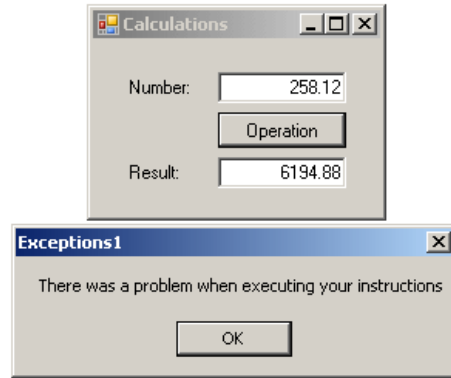
    Result = Number * 24
    txtResult.Text = CStr(Result)

    ThereWasAProblem:
    MsgBox("There was a problem when executing your instructions")
End Sub

```

If you create such a label, you should tell the compiler when to jump to that label. Otherwise, as in this case, the label section would always execute. Here is an example of running the

above version:



In this case, we want the label section to execute only when we want it to. To prevent the compiler from reaching this section if not directed so, you can add an **Exit Sub** line above the label section:

```
Private Sub CalculateClicked(ByVal sender As Object, _
                           ByVal e As EventArgs) _
    Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

    Result = Number * 24
    txtResult.Text = CStr(Result)

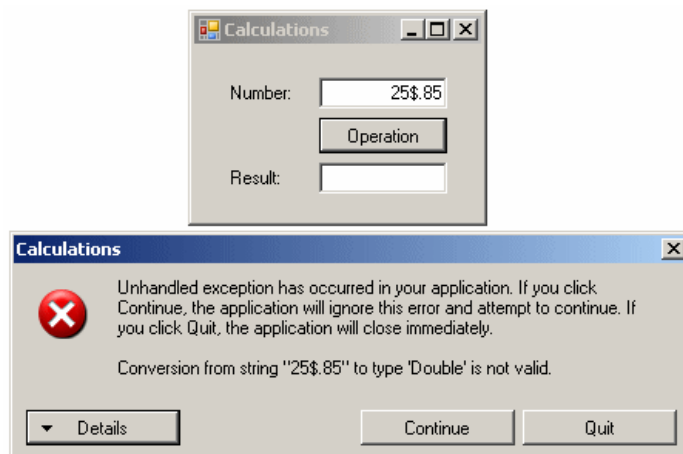
    Exit Sub

ThereWasAProblem:
    MsgBox("There was a problem when executing your instructions")
End Sub
```

This time if you execute the program with an appropriate value, the label section would not be reached.

In Case Of Error, Jump To Label

The above program will compile fine. When executing it, imagine that the user types an inappropriate value such as 25\$.85 instead of 25.85. In this case, the value is not a number, the program would "crash" and let you know that there was a problem:



With some experience, you would know what the problem was, otherwise, you would face a vague explanation. The short story is that the compiler could not continue because, in this case, it could not multiply 25\$.85 by another number.

If a problem occurs when a person is using your program, the compiler may display a nasty and insignificant message to the user who would not know what to do with it. Therefore, you can start by creating an appropriate label as introduced above. An error normally occurs in a function. Therefore, to make your code easier to read, you should create a label that shows that it is made for an error instead of being a regular label. The label should also reflect the name of the function. Here is an example:

```
Private Sub CalculateClicked(ByVal sender As Object, _
```

```

ByVal e As EventArgs) _
Handles btnCalculate.Click

Dim Number As Double
Dim Result As Double

Number = CDbI(txtNumber.Text)

Result = Number * 24
txtResult.Text = CStr(Result)

Exit Sub

btnOperation_Click_Error:
MsgBox("The operation could not be executed", _
MsgBoxStyle.OkOnly, "Operation Error")
End Sub

```

When you think there will be a problem in your code, somewhere in the lines under the name of the function but before the line that could cause the problem, type **On Error GoTo** followed by the name of the label that would deal with the error. Here is an example:

```

Private Sub CalculateClicked(ByVal sender As Object, _
ByVal e As EventArgs) _
Handles btnCalculate.Click

On Error GoTo btnOperation_Click_Error

Dim Number As Double
Dim Result As Double

Number = CDbI(txtNumber.Text)

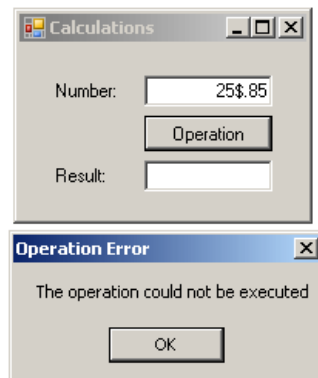
Result = Number * 24
txtResult.Text = CStr(Result)

Exit Sub

btnOperation_Click_Error:
MsgBox("The operation could not be executed", _
MsgBoxStyle.OkOnly, "Operation Error")
End Sub

```

Here is an example of running the program:



When the **On Error GoTo** statement is used, this indicates to the compiler that if any type of error occurs while the code of this function is executed, transfer the compilation to the label. In this case, as soon as something bad happens, the compiler marks the area where the problem occurred, skips the normal code and jumps to the label indicated by the **On Error GoTo** line. After the section of that label is executed, the compiler returns where the error occurred. If there is nothing to solve the problem, the compiler continues down but without executing the lines of code involved. In this case, it would encounter the **Exit Sub** line and get out of the function.

In Case Of Error, Jump To Line

Although the label is more explicit, it only indicates to the compiler what line to jump to in case of a problem. The alternative is to specify a line number instead of a label.

Resume

If a problem occurs in your code and you provide a label to display a friendly message as done above, the compiler would display the message and exit from the function. If this happens, as mentioned above, when the compiler returns where the problem occurred, you can provide an alternative. For example, in our program, if the user provides an inappropriate value that causes the error, you can provide an alternate value and ask the compiler to continue as if

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Goale

nothing happened. In this case, you want to compiler to "resume" its activity.

To indicate that the program should continue, you can use the **Resume** keyword. Here is an example:

```
Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
    Handles btnCalculate.Click
    On Error GoTo btnOperation_Click_Error

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

    Resume

    Result = Number * 24
    txtResult.Text = CStr(Result)

    Exit Sub

btnOperation_Click_Error:
    MsgBox("The operation could not be executed", _
        MsgBoxStyle.OkOnly, "Operation Error")
End Sub
```

When an error occurs, if you want the program to continue with an alternate value than the one that caused the problem, in the label section, type **Resume Next**. Here is an example:

```
Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
    Handles btnCalculate.Click
    On Error GoTo btnOperation_Click_Error

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

    Result = Number * 24
    txtResult.Text = CStr(Result)

    Exit Sub

btnOperation_Click_Error:
    MsgBox("The operation could not be executed", _
        MsgBoxStyle.OkOnly, "Operation Error")
    Resume Next
End Sub
```

In this case, since any numeric variable is initialized with 0, when the compiler returns to the line of code that caused the problem, it would use 0 as a substitute to the inappropriate value. Based on this, you can provide a new value to use in case of error. Here is an example:

```
Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
    Handles btnCalculate.Click
    On Error GoTo btnOperation_Click_Error

    Dim Number As Double
    Dim Result As Double

    Number = CDb1(txtNumber.Text)

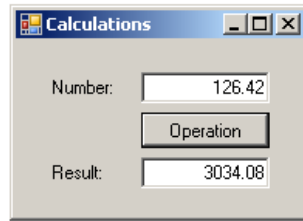
    Result = Number * 24
    txtResult.Text = CStr(Result)

    Exit Sub

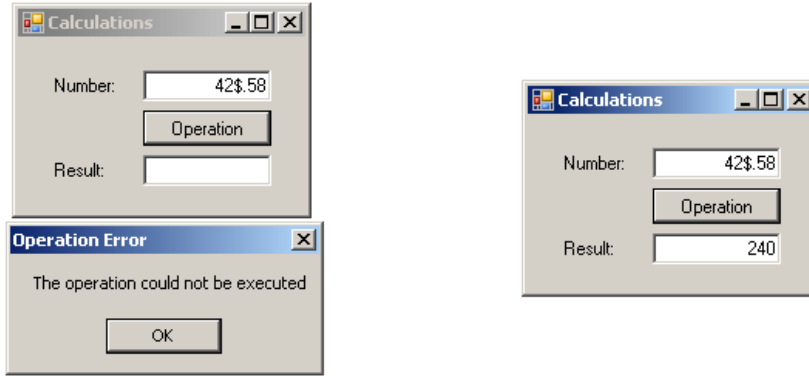
btnOperation_Click_Error:
    MsgBox("The operation could not be executed", _
        MsgBoxStyle.OkOnly, "Operation Error")

    Number = 10
    Resume Next
End Sub
```

Here is one example of running the program:



Here is another example of running the same program:



The Err Object

To support error handling, the Visual Basic library provides a global variable named **Err**. This allows you to identify the error and its description. Because an error depends on what caused it and why, the values of the **Err** variable also depend and are not always the same.

Unlimited Webspace and Unlimited Bandwidth **FREE** Domain Registration **Only Rs. 2000/-**

<ul style="list-style-type: none"> ⇒ 100 Email Ids of 10 GB ⇒ All Database FREE ⇒ 5000 Templates / Website Builder 	<ul style="list-style-type: none"> ⇒ Full Control Panel ⇒ 200 SEO Tools/Softwares/Weblinks
---	--

www.manashosting.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



Exception Handling Fundamentals

Introduction

As opposed to the traditional techniques used to deal with errors, Visual Basic now supports a technique referred to as exception handling. This technique was mostly used by other languages such as C/C++, Object Pascal, C#, etc. This technique is also referred to as structured exception handling (SEH). The **On Error GoTo** system of dealing with errors is referred to as unstructured exception handling (we will abbreviate is unSEH). There were many concerns with unSEH (mostly lastly used in Microsoft Visual Basic 6):

[Java Code Coverage](#)

Jtest Automates JUnit Test Creation & Execution; Checks 500+ Java Rules
www.parasoft.com

[Bulk SMS - Stock Brokers](#)

Stock Alert Mangement system -Best Prices - Pay only 4 delivered SMS

www.Planet41.com

[Windows 2000 Security](#)

Catch hackers red-handed with GFI EventsManager. Free trial!

www.gfi.com

[Market News](#)

Read Experts Advice on the Current Volatile Nature of Stock Market

BusinessToday.DigitalToday.in



Ads by Google

- In unSEH, you have to remember to include **On Error GoTo** sometimes in a random area inside the function. Besides **On Error GoTo**, you have to remember to create a label section. The name of the label has to be faithfully included in the **On Error GoTo** line
- When using the **On Error GoTo** technique, you have to make sure you get out of the function at the right time, which is done using an **Exit Sub** line. If you create this **Exit Sub** line in the wrong area in your function, either the whole code would not be considered or an desired section of the function would still execute, possibly producing an unpredictable result. In the same way, a bizarre way of creating a **Resume** or a **Resume Next** line would prevent the compiler from reaching or considering some sections of a function
- As mentioned previously, the **On Error GoTo** system provides a global variable named **Err**. As flexible as good intentioned as it may appear, to effectively use **Err**, you have to proceed by trial and error because **Err** identifies errors by a number but its documentation does not possibly provide a list of error numbers. This is because **Err** not only depends on the application (Visual Basic, VBA, etc) but also it depends on the circumstance in which it is invoked. The only possible solution is to cause an error in your code in order to cause **Err** to show the numeric error that occurred. Then you have to use that error number and create a custom error message

Because of these uncertainties, you should abandon the **On Error GoTo** traditional error handling and use SEH in all of your new code. Because SEH and unSEH techniques are inherently different, you cannot use both in the same function.

❖ Practical Learning: Introducing Exception Handling

1. Start Microsoft Visual Basic if necessary.
Create a new Windows Application named **GeorgetownCleaningServices1**
2. Design the form as follows:

Control	Name	Text	Additional Properties
Form			Size: 378, 408
Label		Customer Name:	
TextBox	txtCustomerName1		
Label		mm	
Label		dd	
Label		yyyy	
Label		Order Date:	
TextBox	txtMM	1	
TextBox	txtDD	1	
TextBox	txtYYYY	2000	
Label		Item Types	
Label		Qty	
Label		Unit Price	
Label		Sub-Total	
Label		Shirts	
TextBox	txtQtyShirts	0	
TextBox	txtUnitPriceShirts	1.15	
TextBox	txtSubTotalShirts	0.00	
Label		Pants	
TextBox	txtQtyPants	0	
TextBox	txtUnitPricePants	1.95	
TextBox	txtSubTotalPants	0.00	
Label		Other	
TextBox	txtQtyOther	0	
TextBox	txtUnitPriceOther	3.50	
TextBox	txtSubTotalOther	0.00	
Button	btnProcess	Process	
Label		Customer Name:	
TextBox	txtCustomerName2		
Label		Order date:	
TextBox	txtOrderDate		
Label		Tax Rate:	
TextBox	txtTaxRate	5.75	
Label		%	

Button	btnTax	Tax	
Label		Total Order:	
TextBox	txtTotalOrder	0.00	
Label		Tax Amount:	
TextBox	txtTaxAmount	0.00	
Label		Net Price:	
TextBox	txtNetPrice	0.00	
Label		Amount Tended:	
TextBox	txtAmountTended	0.00	
Button	btnDifference	Diff	
Label		Difference:	
TextBox	txtDifference	0.00	

- To arrange the tab sequence, on the main menu, click View -> Tab Order
- On the form, click only the following controls whose squares have a white background, in the indicated order:

The screenshot shows a Windows application window titled "Georgetown Cleaning Services". The form contains various input fields and buttons, each with a small numbered square next to it, indicating the tab order for navigation. The numbered controls are:

- 0: Customer Name text box
- 1: Order Date text box
- 2: Order Date spinner (Day)
- 3: Order Date spinner (Month)
- 4: Order Date spinner (Year)
- 4: Item Types text box
- 5: Quantity text box
- 6: Price text box
- 7: Total text box
- 8: Rate text box
- 9: Tax text box
- 10: Process button
- 11: Amount text box
- 12: Price text box
- 13: Amount Tended text box
- 14: Diff button
- 15: Difference text box
- 16: Customer Name text box
- 17: Order Date text box
- 18: Rate text box
- 19: Tax text box
- 20: Total Order text box
- 21: Amount text box
- 22: Price text box
- 23: Amount Tended text box
- 24: Diff button
- 25: Difference text box
- 26: Amount Tended text box
- 27: Price text box
- 28: Amount Tended text box
- 29: Diff button
- 30: Difference text box
- 31: Amount Tended text box
- 32: Price text box
- 33: Amount Tended text box
- 34: Diff button
- 35: Difference text box
- 36: Amount Tended text box
- 37: Price text box
- 38: Amount Tended text box
- 39: Diff button
- 40: Difference text box
- 41: Amount Tended text box
- 42: Price text box
- 43: Amount Tended text box

- Press Esc
- Right-click the form and click View Code
- Declare a few variables as follows:

```
Public Class Form1

    ' Order Information
    Dim CustomerName As String
    Dim mm As String
    Dim dd As String
    Dim yyyy As String

    ' Quantities of items
    Dim NumberOfShirts As Integer
    Dim NumberOfPants As Integer
    Dim NumberOfOther As Integer

    ' Price of items
    Dim PriceOneShirt As Double
    Dim PriceAPairOfPants As Double
    Dim PriceOther As Double

    ' Each of these sub totals will be used for cleaning items
    Dim SubTotalShirts As Double
    Dim SubTotalPants As Double
    Dim SubTotalOther As Double

    ' Values used to process an order
    Dim TaxRate As Double
```

```

Dim TotalOrder As Double
Dim TaxAmount As Double
Dim SalesTotal As Double

```

```
End Class
```

8. In the Class Name combo box, select btnProcess
9. In the Method Name, select Click and implement its event as follows:

```

Private Sub btnProcess_Click(ByVal sender As Object, _
                             ByVal e As System.EventArgs) _
    Handles btnProcess.Click
    If btnProcess.Text = "Process" Then
        Height = 408
        btnProcess.Text = "Reset"
    Else
        Height = 232
        txtCustomerName1.Text = ""
        txtMM.Text = "1"
        txtDD.Text = "1"
        txtYYYY.Text = "2000"
        txtQtyShirts.Text = "0"
        txtQtyPants.Text = "0"
        txtQtyOther.Text = "0"
        txtSubTotalShirts.Text = "0.00"
        txtSubTotalPants.Text = "0.00"
        txtSubTotalOther.Text = "0.00"

        btnProcess.Text = "Process"
    End If

    ' Request order information from the user
    CustomerName = txtCustomerName1.Text
    mm = txtMM.Text
    dd = txtDD.Text
    yyyy = txtYYYY.Text

    ' Request the quantity of each category of items
    ' Number of Shirts
    NumberOfShirts = CInt(txtQtyShirts.Text)
    ' Number of Pants
    NumberOfPants = CInt(txtQtyPants.Text)
    ' Number of Dresses
    NumberOfOther = CInt(txtQtyOther.Text)

    ' Unit Prices of items
    PriceOneShirt = CDb1(txtUnitPriceShirts.Text)
    PriceAPairOfPants = CDb1(txtUnitPricePants.Text)
    PriceOther = CDb1(txtUnitPriceOther.Text)

    ' Perform the necessary calculations
    SubTotalShirts = NumberOfShirts * PriceOneShirt
    SubTotalPants = NumberOfPants * PriceAPairOfPants
    SubTotalOther = NumberOfOther * PriceOther

    txtSubTotalShirts.Text = CStr(SubTotalShirts)
    txtSubTotalPants.Text = CStr(SubTotalPants)
    txtSubTotalOther.Text = CStr(SubTotalOther)

    ' Calculate the "temporary" total of the order
    TotalOrder = SubTotalShirts + SubTotalPants + SubTotalOther

    ' Display the receipt
    txtCustomerName2.Text = CustomerName
    txtOrderDate.Text = mm + "/" & dd + "/" & yyyy
    txtTotalOrder.Text = CStr(TotalOrder)
End Sub

```

10. In the Class Name combo box, select btnTax
11. In the Method Name, select Click and implement its event as follows:

```

Private Sub btnTax_Click(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
    Handles btnTax.Click
    ' Get the tax rate
    TaxRate = CDb1(txtTaxRate.Text) / 100

    ' Calculate the tax amount using a constant rate
    TaxAmount = TotalOrder * TaxRate
    ' Add the tax amount to the total order
    SalesTotal = TotalOrder + TaxAmount

```

```

        txtTaxAmount.Text = TaxAmount.ToString()
        txtNetPrice.Text = CStr(SalesTotal)
    End Sub

```

12. In the Class Name combo box, select btnDifference
13. In the Method Name, select Click and implement its event as follows:

```

Private Sub btnDifference_Click(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
    Handles btnDifference.Click
    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

    ' Request money for the order
    AmountTended = Cdbl(txtAmountTended.Text)

    ' Calculate the difference owed to the customer
    ' or that the customer still owes to the store
    Difference = AmountTended - SalesTotal

    txtDifference.Text = CStr(Difference)
End Sub

```

14. Return to the form and resize it form to appear as follows:

Item Types	Qty	Unit Price	Sub-Total
Shirts	0	1.15	0.00
Pants	0	1.95	0.00
Other	0	3.50	0.00

15. To execute the application, on the Standard toolbar, click the Start Without Debugging button

Item Types	Qty	Unit Price	Sub-Total
Shirts	7	1.15	0.00
Pants	5	1.95	0.00
Other	2	3.50	0.00

Ads by Google

16. Close the form and return to your programming environment
17. Execute the application again. This time, type a letter such as d for the quantity of shirts and click Process

RadarCube
ASP.NET
OLAP

Native ASP.NET
control for MS AS
based BI solutions

radar-soft.com

18. Click Quit to close the form and return to your programming environment

Try to Catch the Error

As mentioned already, errors are likely going to occur in your program. The more you anticipate them and take action, the better your application can be. We have already seen that syntax errors are usually human mistakes such as misspelling, bad formulation of expressions, etc. The compiler will usually help you fix the problem by pointing it out.

SEH is based on two main keywords: **Try** and **Catch**. An exception handling section starts with the **Try** keyword and stops with the **End Try** statement. Between **Try** and **End Try**, there must be at least one **Catch** section. Therefore, exception handling uses the following formula:

```

Try
    ' Code to execute in case everything is alright
Catch
    ' If something bad happened, deal with it here
End Try

```

Exception handling always starts with the **Try** keyword. Under the **Try** line, Write the normal code that the compiler must execute. Here is an example:

```

Private Sub CalculateClicked(ByVal sender As Object, _
                           ByVal e As EventArgs) _
    Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Try
        Number = CDb1(TextBox("Enter a number:"))
    End Try

End Sub

```

As the compiler is treating code in the **Try** section, if it encounters a problem, it "gets out" of the **Try** section and starts looking for a **Catch** section. Therefore, you MUST always have a **Catch** section. If you do not, the program will not compile. A **Catch** section must be written before the **End Try** line:

```

Private Sub CalculateClicked(ByVal sender As Object, _
                           ByVal e As EventArgs) _
    Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Try
        Number = CDb1(TextBox("Enter a number:"))
    Catch
    End Try

End Sub

```

When the **Catch** keyword is simply written as above, it would be asked to treat any error that occurs. For example, if you execute the above code with a number such as 35\$.75 instead of 35\$.75, nothing would appear to happen. This would indicate that the error was found and vaguely dealt with. One problem in this case is that the compiler would not bother to let the user know why there is no result displayed. Because there can be various types of errors in a program, you also should make your program more intuitive and friendlier so that, when an error occurs, the user would know the type of problem. This is also useful if somebody calls you and says that your program is not functioning right. If there is a way the user can tell you what exact type of error is displaying, maybe you would find the solution faster.

❖ Practical Learning: Catching Exceptions

1. To introduce exceptions, access the form's code and change the events of the buttons as follows:

```

Public Class Form1

    . . . No Change

    Private Sub btnProcess_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
    Handles btnProcess.Click

        . . . No Change

        ' Request the quantity of each category of items
        ' Number of Shirts
    Try
        NumberOfShirts = CInt(txtQtyShirts.Text)
    Catch

    End Try

        ' Number of Pants
    Try
        NumberOfPants = CInt(txtQtyPants.Text)
    Catch

    End Try

        ' Number of Dresses
    Try

```

```

        NumberOfOther = CInt(txtQtyOther.Text)
    Catch

    End Try

    ' Unit Prices of items
    Try
        PriceOneShirt = CDbI(txtUnitPriceShirts.Text)
    Catch

    End Try

    Try
        PriceAPairOfPants = CDbI(txtUnitPricePants.Text)
    Catch

    End Try

    Try
        PriceOther = CDbI(txtUnitPriceOther.Text)
    Catch

    End Try

    . . . No Change
End Sub

Private Sub btnTax_Click(ByVal sender As Object, _
                        ByVal e As System.EventArgs) _
    Handles btnTax.Click
    ' Get the tax rate
    Try
        TaxRate = CDbI(txtTaxRate.Text) / 100
    Catch

    End Try
    ' Calculate the tax amount using a constant rate
    TaxAmount = TotalOrder * TaxRate
    ' Add the tax amount to the total order
    SalesTotal = TotalOrder + TaxAmount

    txtTaxAmount.Text = TaxAmount.ToString()
    txtNetPrice.Text = SalesTotal.ToString()
End Sub

Private Sub btnDifference_Click(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
    Handles btnDifference.Click
    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

    ' Request money for the order
    Try
        AmountTended = CDbI(txtAmountTended.Text)
    Catch

    End Try

    ' Calculate the difference owed to the customer
    ' or that the customer still owes to the store
    Difference = AmountTended - SalesTotal

    txtDifference.Text = CStr(Difference)
End Sub
End Class

```

2. Execute the application. This time, type invalid values in the quantity text boxes and other text boxes where the user is supposed to enter some values
3. Click Process

Item Types	Qty	Unit Price	Sub-Total
Shirts	2	1.15	2.30
Pants	W	1.95	0.00
Other	0	3.50	0.00

4. Return to your programming environment

The Error Message

As mentioned already, if an error occurs when processing the program in the **Try** section, the compiler transfers the processing to the next **Catch** section. You can then use the catch section to deal with the error. At a minimum, you can display a message to inform the user. To do this, you can create a message box in the **Catch** section. Here is an example:

```
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

        Private lblNumber As Label
        Private txtNumber As TextBox
        Friend WithEvents btnCalculate As Button
        Private lblResult As Label
        Private txtResult As TextBox

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            Text = "Exception Behavior"

            lblNumber = New Label
            lblNumber.Location = New Point(17, 23)
            lblNumber.Text = "Number:"
            lblNumber.AutoSize = True

            txtNumber = New TextBox
            txtNumber.Location = New Point(78, 20)
            txtNumber.Size = New Size(83, 20)

            btnCalculate = New Button
            btnCalculate.Location = New Point(78, 45)
            btnCalculate.Text = "Calculate"
            btnCalculate.Size = New Size(83, 23)

            lblResult = New Label
            lblResult.Location = New Point(17, 75)
            lblResult.Text = "Result:"
            lblResult.AutoSize = True

            txtResult = New TextBox
            txtResult.Location = New Point(76, 72)
            txtResult.Size = New Size(83, 20)
        End Sub
    End Class
End Module
```

```

Controls.Add(lblNumber)
Controls.Add(txtNumber)
Controls.Add(btnCalculate)
Controls.Add(lblResult)
Controls.Add(txtResult)

End Sub

Private Sub CalculateClicked(ByVal sender As Object, _
                            ByVal e As EventArgs) _
    Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Try
        Number = CDb1(txtNumber.Text)
        Result = Number * 12.48
        txtResult.Text = CStr(Result)
    Catch
        MsgBox("Something bad happened")
    End Try
End Sub
End Class

Function Main() As Integer

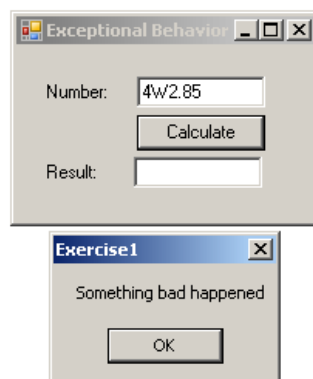
    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function

End Module

```



Of course, your message may not be particularly clear but this time, the program will not crash.

❖ Practical Learning: Displaying Custom Messages

1. To display custom messages to the user, change the code as follows:

```

Public Class Form1

    . . . No Change

    Private Sub btnProcess_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
        Handles btnProcess.Click

        . . . No Change

        ' Request the quantity of each category of items
        ' Number of Shirts
    Try
        NumberOfShirts = CInt(txtQtyShirts.Text)
    Catch
        MsgBox("The value you typed for the number of " & _
              "shirts is not a valid number." & _
              vbCrLf & "Please enter a natural number such " & _
              "as 2 or 24 or even 248")
    End Try

    ' Number of Pants
    Try

```

```

        NumberOfPants = CInt(txtQtyPants.Text)
    Catch
        MsgBox("The value you typed for the number of " & _
            "pair or pants is not a valid number." & _
            vbCrLf & "Please enter a natural number such " & _
            "as 2 or 24 or even 248")
    End Try

    ' Number of other items
    Try
        NumberOfOther = CInt(txtQtyOther.Text)
    Catch
        MsgBox("The value you typed for the number of " & _
            "other items is not a valid number." & _
            vbCrLf & "Please enter a natural number such " & _
            "as 2 or 24 or even 248")
    End Try

    ' Unit Prices of items
    Try
        PriceOneShirt = CDbI(txtUnitPriceShirts.Text)
    Catch
        MsgBox("The value you entered for the unit price " & _
            "of a shirt is not a recognizable currency " & _
            "amount." & vbCrLf & _
            "Only natural or decimal numbers " & _
            "are allowed. Please consult the management " & _
            "to know the valid prices.")
    End Try

    Try
        PriceAPairOfPants = CDbI(txtUnitPricePants.Text)
    Catch
        MsgBox("The value you entered for the unit price of " & _
            "a pair of pants is not a recognizable " & _
            "currency amount." & vbCrLf & _
            "Only natural or decimal " & _
            "numbers are allowed. You can consult the " & _
            "management to find out about " & _
            "the allowable prices.")
    End Try

    Try
        PriceOther = CDbI(txtUnitPriceOther.Text)
    Catch
        MsgBox("The value you entered for the unit " & _
            "price of other items is not a valid amount." & _
            vbCrLf & "You must enter only a natural or a " & _
            "decimal number. For more information, " & _
            "please consult the management to get " & _
            "the right prices.")
    End Try

    . . . No Change
End Sub

Private Sub btnTax_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnTax.Click
    ' Get the tax rate
    Try
        TaxRate = CDbI(txtTaxRate.Text) / 100
    Catch
        MsgBox("The value you entered is not " & _
            "recognized as a valid tax rate." & _
            vbCrLf & "A valid tax rate is a value " & _
            "between 0 and 100.00" & _
            vbCrLf & "Please try again.")
    End Try
    ' Calculate the tax amount using a constant rate
    TaxAmount = TotalOrder * TaxRate
    ' Add the tax amount to the total order
    SalesTotal = TotalOrder + TaxAmount

    txtTaxAmount.Text = TaxAmount.ToString()
    txtNetPrice.Text = SalesTotal.ToString()
End Sub

Private Sub btnDifference_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnDifference.Click
    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

```

```

' Request money for the order
Try
    AmountTended = CDb1(txtAmountTended.Text)
Catch
    MsgBox("The value you entered for the amount " & _
           "tended is not valid. Only natural or " & _
           "decimal numbers are allowed." & _
           "Please try again.")

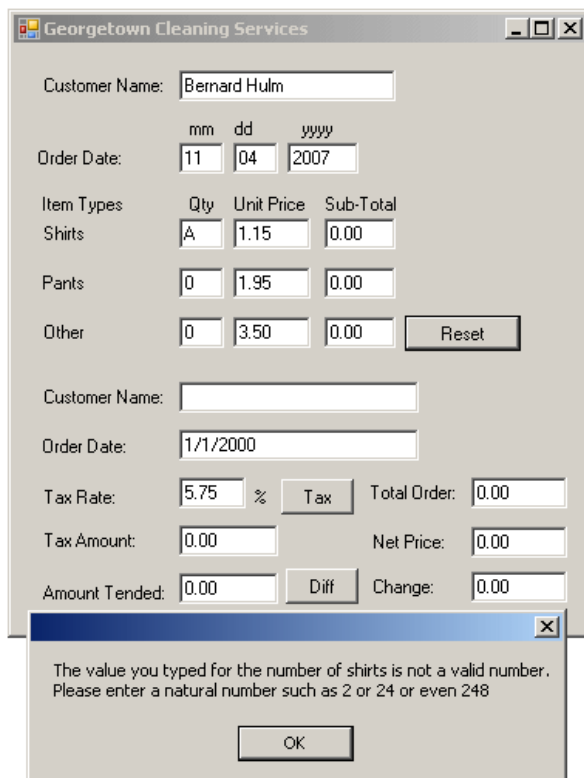
End Try

' Calculate the difference owed to the customer
' or that the customer still owes to the store
Difference = AmountTended - SalesTotal

txtDifference.Text = CStr(Difference)
End Sub
End Class

```

2. Test the application with valid and invalid values. Here is an example:



3. Return to Notepad

Unlimited Webspace and Unlimited Bandwidth		FREE	Only Rs. 2000/-
⇒ 100 Email Ids of 10 GB ⇒ All Database FREE ⇒ 5000 Templates / Website Builder		Domain Registration ⇒ Full Control Panel ⇒ 200 SEO Tools/Softwares/Weblinks	
www.manashosting.com		Feedback - Ads by Google	

[Home](#)

Copyright © 2008 FunctionX, Inc.



A Review of .NET Exception Classes

Introduction

The .NET Framework provides various classes to handle almost any type of exception you can think of. There are so many of these classes that we can only mention a few.

[PDF Display and Printing](#)

ABCpdf PDF rendering & PDF generation, for C#, VB, ASP & .NET
www.websupergoo.com

There are two main ways you can use one of the classes of the .NET Framework. If you know for sure that a particular exception will be produced, pass its name to a **Catch** clause. Then, in the **Catch** section, display a custom message. The second option you have consists of using the **Throw** keyword. We will study it later.



Ads by Google

In most cases, we will try to always indicate the type of exception that could be thrown if something goes wrong in a program.

The Format Exception

Everything the user types into an application using the keyboard is primarily a string and you must convert it to the appropriate type before using it. When you request a specific .NET type of value from the user, after the user has typed it and you decide to convert it to the appropriate type, if your conversion fails, the program produces (we will use the word "throw") an error. The error is of from the **FormatException** class.

Here is a program that deals with a **FormatException** exception:

```
Imports System.Drawing
Imports System.Windows.Forms

Module Exercise

    Public Class Starter
        Inherits Form

        Private lblNumber As Label
        Private txtNumber As TextBox
        Friend WithEvents btnCalculate As Button
        Private lblResult As Label
        Private txtResult As TextBox

        Dim components As System.ComponentModel.Container

        Public Sub New()
            InitializeComponent()
        End Sub

        Public Sub InitializeComponent()
            Text = "Exceptional Behavior"

            lblNumber = New Label
            lblNumber.Location = New Point(17, 23)
            lblNumber.Text = "Number:"
            lblNumber.AutoSize = True

            txtNumber = New TextBox
            txtNumber.Location = New Point(78, 20)
            txtNumber.Size = New Size(83, 20)

            btnCalculate = New Button
            btnCalculate.Location = New Point(78, 45)
            btnCalculate.Text = "Calculate"
            btnCalculate.Size = New Size(83, 23)

            lblResult = New Label
```

```

lblResult.Location = New Point(17, 75)
lblResult.Text = "Result:"
lblResult.AutoSize = True

txtResult = New TextBox
txtResult.Location = New Point(76, 72)
txtResult.Size = New Size(83, 20)

Controls.Add(lblNumber)
Controls.Add(txtNumber)
Controls.Add(btnCalculate)
Controls.Add(lblResult)
Controls.Add(txtResult)

End Sub

Private Sub CalculateClicked(ByVal sender As Object, _
                           ByVal e As EventArgs) _
    Handles btnCalculate.Click

    Dim Number As Double
    Dim Result As Double

    Try
        Number = Double.Parse(txtNumber.Text)
        Result = Number * 12.48
        txtResult.Text = CStr(Result)
    Catch ex As FormatException
        MsgBox("Invalid Value!")
    End Try

End Sub
End Class

Function Main() As Integer

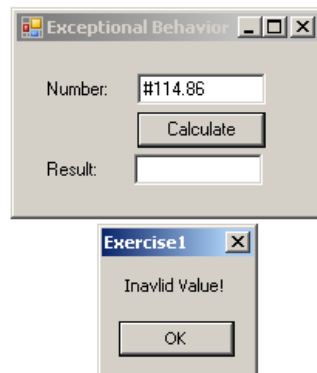
    Dim frmStart As Starter = New Starter

    Application.Run(frmStart)

    Return 0
End Function

End Module

```



❖ Practical Learning: Using the FormatException Class

1. Open the GeorgetownCleaningServices1 application from the previous lesson
2. Change the code as follows:

```

Public Class Form1
    . . . No Change

    Private Sub btnProcess_Click(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
        Handles btnProcess.Click
        . . . No Change

        ' Request the quantity of each category of items
        ' Number of Shirts
        Try
            NumberOfShirts = CInt(txtQtyShirts.Text)
        Catch ex As FormatException
            MsgBox("The value you typed for the number of " & _
                "shirts is not a valid number." & _

```

```

        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")
    End Try

    ' Number of Pants
    Try
        NumberOfPants = CInt(txtQtyPants.Text)
    Catch ex As FormatException
        MsgBox("The value you typed for the number of " & _
            "pair or pants is not a valid number." & _
            vbCrLf & "Please enter a natural number such " & _
            "as 2 or 24 or even 248")
    End Try

    ' Number of other items
    Try
        NumberOfOther = CInt(txtQtyOther.Text)
    Catch ex As FormatException
        MsgBox("The value you typed for the number of " & _
            "other items is not a valid number." & _
            vbCrLf & "Please enter a natural number such " & _
            "as 2 or 24 or even 248")
    End Try

    ' Unit Prices of items
    Try
        PriceOneShirt = CDbI(txtUnitPriceShirts.Text)
    Catch ex As FormatException
        MsgBox("The value you entered for the unit price " & _
            "of a shirt is not a recognizable currency " & _
            "amount." & vbCrLf & _
            "Only natural or decimal numbers " & _
            "are allowed. Please consult the management " & _
            "to know the valid prices.")
    End Try

    Try
        PriceAPairOfPants = CDbI(txtUnitPricePants.Text)
    Catch ex As FormatException
        MsgBox("The value you entered for the unit price of " & _
            "a pair of pants is not a recognizable " & _
            "currency amount." & vbCrLf & _
            "Only natural or decimal " & _
            "numbers are allowed. You can consult the " & _
            "management to find out about " & _
            "the allowable prices.")
    End Try

    Try
        PriceOther = CDbI(txtUnitPriceOther.Text)
    Catch ex As FormatException
        MsgBox("The value you entered for the unit " & _
            "price of other items is not a valid amount." & _
            vbCrLf & "You must enter only a natural or a " & _
            "decimal number. For more information, " & _
            "please consult the management to get " & _
            "the right prices.")
    End Try

    . . . No Change
End Sub

Private Sub btnTax_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnTax.Click

    ' Get the tax rate
    Try
        TaxRate = CDbI(txtTaxRate.Text) / 100
    Catch ex As FormatException
        MsgBox("The value you entered is not " & _
            "recognized as a valid tax rate." & _
            vbCrLf & "A valid tax rate is a value " & _
            "between 0 and 100.00" & _
            vbCrLf & "Please try again.")
    End Try

    ' Calculate the tax amount using a constant rate
    TaxAmount = TotalOrder * TaxRate
    ' Add the tax amount to the total order
    SalesTotal = TotalOrder + TaxAmount

    txtTaxAmount.Text = TaxAmount.ToString()
    txtNetPrice.Text = SalesTotal.ToString()
End Sub

```

```

Private Sub btnDifference_Click(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
    Handles btnDifference.Click
    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

    ' Request money for the order
    Try
        AmountTended = CDb1(txtAmountTended.Text)
    Catch ex As FormatException
        MsgBox("The value you entered for the amount " & _
              "tended is not valid. Only natural or " & _
              "decimal numbers are allowed." & _
              "Please try again.")
    End Try

    ' Calculate the difference owed to the customer
    ' or that the customer still owes to the store
    Difference = AmountTended - SalesTotal

    txtDifference.Text = CStr(Difference)
End Sub
End Class

```

[Ads by Google](#)

[RadarCube](#)
[ASP.NET](#)
[OLAP](#)

- Execute the application and return to your programming environment

The Overflow Exception

A computer application receives, processes, and produces values on a regular basis as the program is running. To better manage these values, as we saw when studying variables and data types, the compiler uses appropriate amounts of space to store its values. It is not unusual that either you the programmer or a user of your application provides a value that is beyond the allowed range based on the data type. For example, a byte uses 8 bits to store a value and a combination of 8 bits can store a number no more than 255. If you provide a value higher than 255 to be stored in a byte, you get an error. Consider the following program:

Native ASP.NET
control for MS AS
based BI solutions

[radar-soft.com](#)

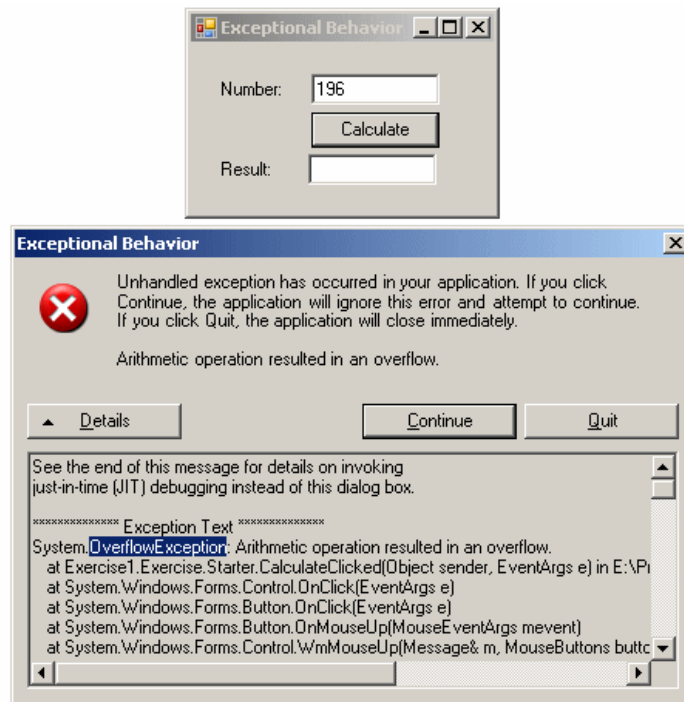
```

Private Sub CalculateClicked(ByVal sender As Object, _
                             ByVal e As EventArgs) _
    Handles btnCalculate.Click
    Dim Number As Byte
    Dim Result As Byte

    Try
        Number = Byte.Parse(txtNumber.Text)
        Result = Number * 12
        txtResult.Text = CStr(Result)
    Catch ex As FormatException
        MsgBox("Invalid Value!")
    End Try
End Sub

```

When a value beyond the allowable range is asked to be stored in memory, the compiler produces (the verb is "throws" as we will learn soon) an error of the **OverflowException** class. Here is an example of running the program with a bad number:



As with the other errors, when this exception is thrown, you should take appropriate action.

The Argument Out of Range Exception

Once again, in a .NET Framework application, a value is passed to the **Parse()** method of its data type for analysis. For a primitive data type, the **Parse()** method scans the string and if the converted value is beyond a determined range, the compiler throws an **ArgumentOutOfRangeException** exception.

❖ Practical Learning: Using an ArgumentOutOfRangeException Exception

- Under the CustomerName variable, declare a variable named OrderTime of type DateTime

```
Public Class Form1
    ' Order Information
    Dim CustomerName As String
    Dim OrderDate As DateTime
    Dim mm As String
    Dim dd As String
    Dim yyyy As String
```

- Change the Click event of the Process button as follows:

```
Private Sub btnProcess_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnProcess.Click
    . . . No Change

    ' Request order information from the user
    CustomerName = txtCustomerName1.Text

    Try
        Dim mm As Integer, dd As Integer, yyyy As Integer
        mm = Integer.Parse(txtMM.Text)
        dd = Integer.Parse(txtDD.Text)
        yyyy = Integer.Parse(txtYYYY.Text)
        OrderDate = New DateTime(yyyy, mm, dd)

    Catch ex As ArgumentOutOfRangeException
        MsgBox("The date you entered is not valid" & _
            "- Please try again!")
    End Try

    . . . No Change

    ' Display the receipt
    txtCustomerName2.Text = CustomerName
    txtOrderDate.Text = FormatDateTime(OrderDate, DateFormat.LongDate)
```

```
txtTotalOrder.Text = CStr(TotalOrder)
End Sub
```

- Execute the application.
- To test it enter valid and invalid values for the controls. Here is an example:

- Close the form and return to your programming environment

The Divide by Zero Exception

Division by zero is an operation to always avoid. It is so important that it is one of the most fundamental exceptions of the computer. It is addressed at the core level even by the processors. It is also addressed by the operating systems at their level. It is also addressed by most, if not all, compilers. It is also addressed by most, if not, all libraries. This means that this exception is never welcomed anywhere. The .NET Framework also provides its own class to face this operation.

If an attempt to divide a value by 0, the compiler throws a **DivideByZeroException** exception.

Unlimited Webspace and Unlimited Bandwidth **FREE** Domain Registration **Only Rs. 2000/-**

⇒ 100 Email Ids of 10 GB ⇒ Full Control Panel

⇒ All Database FREE ⇒ 200 SEO Tools/Softwares/Weblinks

⇒ 5000 Templates / Website Builder

www.manashosting.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Techniques of Using Exceptions

Throwing an Exception

Ads by Google

As mentioned above, the **Exception** class is equipped with a **Message** property that carries a message for the error that occurred. We also mentioned that the message of this property may not be particularly useful to a user. Fortunately, you can create your own message and pass it to the **Exception** object. To be able to receive custom messages, the **Exception** class provides the following constructor:

```
Public Sub New(message As String)
```

Compare Excel tables

Powerful and handy add-on for Excel 2000-2007 files comparison.

www.office-excel.com

To use it, in the section where you are anticipating the error, type the **Throw** keyword followed by a **New** instance of the **Exception** class using the constructor that takes a string. Here is an example:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/") Then
            Throw New Exception(Oper)
        End If

        Select Case Oper
            Case "+"
                Result = Operand1 + Operand2

            Case "-"
                Result = Operand1 - Operand2

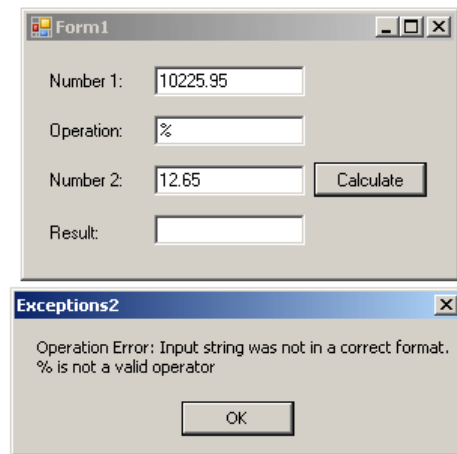
            Case "*"
                Result = Operand1 * Operand2

            Case "/"
                Result = Operand1 / Operand2

            Case Else
                MsgBox("Bad Operation")
        End Select

        TextBox4.Text = CStr(Result)

    Catch Ex As Exception
        MsgBox("Operation Error: " & Ex.Message & _
            vbCrLf & Oper & " is not a valid operator")
    End Try
End Sub
```



Catching Various Exceptions

In the above examples, when we anticipated some type of problem, we instructed the compiler to use our default catch section. We left it up to the compiler to find out when there was a problem and we provided a catch section to deal with it. A method with numerous or complex operations and requests can also produce different types of errors. With such a type of program, you should be able to face different problems and deal with them individually, each by its own kind. To do this, you can create different catch sections, each made for a particular error. The formula used would be:

```
Try
    ' Code to Try
Catch Arg1
    ' One Exception
Catch Arg2
    ' Another Exception
End Try
```

The compiler would proceed in a top-down:

1. Following the normal flow of the program, the compiler enters the **try** block
2. If no exception occurs in the **Try** block, the rest of the **Try** block is executed
If an exception occurs in the **Try** block, the compiler registers the type of error that occurred. If there is a **Throw** line, the compiler registers it also:
 - a. The compiler gets out of the **Try** section
 - b. The compiler examines the first **Catch**. If the first **Catch** matches the thrown error, that catch executes and the exception handling routine may seize. If the first **Catch** does not match the thrown error, the compiler proceeds with the next **Catch**
 - c. The compiler checks the next match, if any, and proceeds as in the first match. This continues until the compiler finds a **Catch** clause that matches the thrown error
 - d. If one of the catches matches the thrown error, its body executes. If no **Catch** matches the thrown error, the compiler calls the **Exception** class and uses the default message

Multiple catches are written if or when a try block is expected to throw different types of errors. For example, in our calculator, we want to consider only the addition, the subtraction, the multiplication, and the division. It is also likely that the user may type one or two invalid numbers. This leads us to know that our program can produce at least two types of errors at this time. Based on this, we can address them using two catch clauses as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."
    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)
        If (Oper <> "+") And _
            (Oper <> "-") And _
```

```

        (Oper <> "*" ) And _
        (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If

    Select Case Oper

        Case "+"
            Result = Operand1 + Operand2

        Case "-"
            Result = Operand1 - Operand2

        Case "*"
            Result = Operand1 * Operand2

        Case "/"
            Result = Operand1 / Operand2

        Case Else
            MsgBox("Bad Operation")

    End Select

    TextBox4.Text = CStr(Result)

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch Ex As Exception
    MsgBox("Operation Error: " & Ex.Message & _
        vbCrLf & Oper & " is not a valid operator")
End Try
End Sub

```

This program works fine as long as the user types two valid numbers and a valid arithmetic operator. Anything else, such as an invalid number or an unexpected operator would cause an error to be thrown:

Obviously various bad things could happen when this program is running. Imagine that the user wants to perform a division. You need to tell the compiler what to do if the user enters the denominator as 0 (or 0.00). If this happens, one of the options you should consider is to display a message and get out. Fortunately, the .NET Framework provides the **DivideByZeroException** class to deal with an exception caused by division by zero. As done with the message passed to the **Exception** class, you can compose your own message and pass it to the **DivideByZeroException(string message)** constructor.

Exception is the parent of all exception classes. It corresponds to the type of Catch that takes no argument. Therefore, if you write various catch blocks, the one that either takes nor argument or is of the **Exception** type must be the last.

Here is an example that catches two types of exceptions:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/") Then
            Throw New Exception(Oper)
        End If

        Select Case Oper
            Case "+"
                Result = Operand1 + Operand2

            Case "-"
                Result = Operand1 - Operand2

            Case "*"
                Result = Operand1 * Operand2

            Case "/"
                If Operand2 = 0 Then
                    Throw New DivideByZeroException(_
                        "Division by zero is not allowed")
                End If

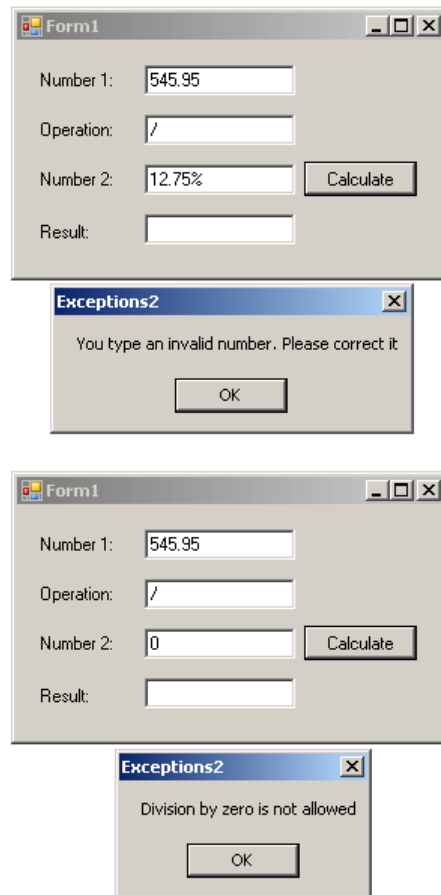
                Result = Operand1 / Operand2

            Case Else
                MsgBox("Bad Operation")

        End Select

        TextBox4.Text = CStr(Result)

    Catch ex As FormatException
        MsgBox("You type an invalid number. Please correct it")
    Catch ex As DivideByZeroException
        MsgBox(ex.Message)
    Catch
        MsgBox("Invalid Operation: " & vbCrLf & _
            Oper & " is not a valid operator")
    End Try
End Sub
```



❖ Practical Learning: Identifying the Thrown Exception

1. To catch various exceptions, change the code as follows:

```
Public Class Form1

    ' Order Information
    Dim CustomerName As String
    Dim OrderDate As DateTime
    Dim mm As String
    Dim dd As String
    Dim yyyy As String

    ' Quantities of items
    Dim NumberOfShirts As Integer
    Dim NumberOfPants As Integer
    Dim NumberOfOther As Integer

    ' Price of items
    Dim PriceOneShirt As Double
    Dim PriceAPairOfPants As Double
    Dim PriceOther As Double

    ' Each of these sub totals will be used for cleaning items
    Dim SubTotalShirts As Double
    Dim SubTotalPants As Double
    Dim SubTotalOther As Double

    ' Values used to process an order
    Dim TaxRate As Double
    Dim TotalOrder As Double
    Dim TaxAmount As Double
    Dim SalesTotal As Double

    Private Sub btnProcess_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnProcess.Click
        If btnProcess.Text = "Process" Then
            Height = 408
            btnProcess.Text = "Reset"
        Else
            Height = 232
            txtCustomerName1.Text = ""
        End If
    End Sub
End Class
```

```

txtMM.Text = "1"
txtDD.Text = "1"
txtYYYY.Text = "2000"
txtQtyShirts.Text = "0"
txtQtyPants.Text = "0"
txtQtyOther.Text = "0"
txtSubTotalShirts.Text = "0.00"
txtSubTotalPants.Text = "0.00"
txtSubTotalOther.Text = "0.00"

btnProcess.Text = "Process"
End If

' Request order information from the user
CustomerName = txtCustomerName1.Text

Try
    Dim mm As Integer, dd As Integer, yyyy As Integer
    mm = Integer.Parse(txtMM.Text)
    dd = Integer.Parse(txtDD.Text)
    yyyy = Integer.Parse(txtYYYY.Text)
    OrderDate = New DateTime(yyyy, mm, dd)

    ' This exception is thrown if the user types a value that cannot
    ' be converted into a valid number
Catch ex As FormatException
    MsgBox("Error: " & ex.Message & _
        vbCrLf & "The value you entered " & _
        "is not a valid number")

    ' This exception is thrown if the
    ' values that user had typed cannot
    ' produce a valid date value
Catch ex As ArgumentOutOfRangeException
    MsgBox("The date you entered is not valid" & _
        "- Please try again!")
End Try

' Request the quantity of each category of items
' Number of Shirts
Try
    NumberOfShirts = CInt(txtQtyShirts.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "shirts is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")

    ' This exception is thrown if the user types a negative value
Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Number of Pants
Try
    NumberOfPants = CInt(txtQtyPants.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "pair or pants is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")
Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Number of other items
Try
    NumberOfOther = CInt(txtQtyOther.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "other items is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")
Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Unit Prices of items
Try
    PriceOneShirt = CDbL(txtUnitPriceShirts.Text)
    If PriceOneShirt < 0 Then
        Throw New Exception("Negative numbers are not allowed " & _
            "for the price of a shirt")
    End If
End Try

```

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google


```

    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit price " & _
        "of a shirt is not a recognizable currency " & _
        "amount." & vbCrLf & _
        "Only natural or decimal numbers " & _
        "are allowed. Please consult the management " & _
        "to know the valid prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

Try
    PriceAPairOfPants = CDbI(txtUnitPricePants.Text)
    If PriceAPairOfPants < 0 Then
        Throw New Exception("Negative numbers are not allowed " & _
            "for the price of a pair of pants")
    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit price of " & _
        "a pair of pants is not a recognizable " & _
        "currency amount." & vbCrLf & _
        "Only natural or decimal " & _
        "numbers are allowed. You can consult the " & _
        "management to find out about " & _
        "the allowable prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

Try
    PriceOther = CDbI(txtUnitPriceOther.Text)
    If PriceOther < 0 Then
        Throw New Exception("Negative numbers are " & _
            "not allowed for the price")
    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit " & _
        "price of other items is not a valid amount." & _
        vbCrLf & "You must enter only a natural or a " & _
        "decimal number. For more information, " & _
        "please consult the management to get " & _
        "the right prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

' Perform the necessary calculations
SubTotalShirts = NumberOfShirts * PriceOneShirt
SubTotalPants = NumberOfPants * PriceAPairOfPants
SubTotalOther = NumberOfOther * PriceOther

txtSubTotalShirts.Text = CStr(SubTotalShirts)
txtSubTotalPants.Text = CStr(SubTotalPants)
txtSubTotalOther.Text = CStr(SubTotalOther)

' Calculate the "temporary" total of the order
TotalOrder = SubTotalShirts + SubTotalPants + SubTotalOther

' Display the receipt
txtCustomerName2.Text = CustomerName
txtOrderDate.Text = FormatDateTime(OrderDate, DateFormat.LongDate)
txtTotalOrder.Text = CStr(TotalOrder)
End Sub

Private Sub btnTax_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnTax.Click

' Get the tax rate
Try
    TaxRate = CDbI(txtTaxRate.Text) / 100
    If TaxRate < 0 Then
        Throw New Exception("Negative numbers are not " & _
            "allowed for a tax rate")
    End If

Catch ex As FormatException
    MsgBox("The value you entered is not " & _
        "recognized as a valid tax rate." & _
        vbCrLf & "A valid tax rate is a value " & _
        "between 0 and 100.00" & _
        vbCrLf & "Please try again.")

```

```

Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

' Calculate the tax amount using a constant rate
TaxAmount = TotalOrder * TaxRate
' Add the tax amount to the total order
SalesTotal = TotalOrder + TaxAmount

txtTaxAmount.Text = TaxAmount.ToString()
txtNetPrice.Text = SalesTotal.ToString()
End Sub

Private Sub btnDifference_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnDifference.Click
    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

    ' Request money for the order
    Try
        AmountTended = CDb1(txtAmountTended.Text)
        If AmountTended < 0 Then
            Throw New Exception("Negative numbers are not " & _
                "allowed forthe amount tended")
        End If

        Catch ex As FormatException
            MsgBox("The value you entered for the amount " & _
                "tended is not valid. Only natural or " & _
                "decimal numbers are allowed." & _
                "Please try again.")
        Catch Ex As Exception
            MsgBox("Something bad happened")
        End Try

        ' Calculate the difference owed to the customer
        ' or that the customer still owes to the store
        Difference = AmountTended - SalesTotal

        txtDifference.Text = CStr(Difference)
    End Sub
End Class

```

2. Test the application and return to your programming environment

Exceptions Nesting

The calculator simulator we have studied so far performs a division as one of its assignments. We learned that, in order to perform any operation, the compiler must first make sure that the user has entered a valid operator. Provided the operator is one of those we are expecting, we also must make sure that the user typed valid numbers. Even if these two criteria are met, it is still possible that the user would enter 0 for the denominator. The block that is used to check for a non-zero denominator depends on the exception that validates the operators. The exception that could result from a zero denominator depends on the user first entering a valid number for the denominator.

You can create an exception inside of another. This is referred to as nesting an exception. This is done by applying the same techniques used to nest conditional statements. This means that you can write an exception that depends on, and is subject to, another exception. To nest an exception, create a **Try** clause in the body of the parent exception. The nested **Try** clause must be followed by its own **Catch** clause(s). To effectively handle the exception, make sure you include an appropriate **Throw** in the **Try** block. Here is an example:

```

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If
    End Try

```

```

Select Case Oper

    Case "+"
        Result = Operand1 + Operand2
        TextBox4.Text = CStr(Result)

    Case "-"
        Result = Operand1 - Operand2
        TextBox4.Text = CStr(Result)

    Case "*"
        Result = Operand1 * Operand2
        TextBox4.Text = CStr(Result)

    Case "/"
        Try
            If Operand2 = 0 Then
                Throw New DivideByZeroException(_
                    "Division by zero is not allowed")
            End If
            Result = Operand1 / Operand2

        Catch ex As DivideByZeroException
            MsgBox(ex.Message)
        End Try

    Case Else
        MsgBox("Bad Operation")

End Select

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch
    MsgBox("Invalid Operation: " & vbCrLf & _
        Oper & " is not a valid operator")
End Try
End Sub

```

Exceptions and Functions

One of the most effective techniques used to deal with code is to isolate assignments in different functions. For example, the **Select Case** statement that was performing the operations in the "normal" version of our program could be written as follows:

```

Private Function Calculate(ByVal Value1 As Double, _
    ByVal Value2 As Double, _
    ByVal symbol As Char) As Double

    Dim Result As Double = 0.0

    Select Case symbol

        Case "+"
            Result = Value1 + Value2

        Case "-"
            Result = Value1 - Value2

        Case "*"
            Result = Value1 * Value2

        Case "/"
            Result = Value1 / Value2
    End Select

    Calculate = Result
End Function

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _

```

```

        (Oper <> "-") And _
        (Oper <> "**") And _
        (Oper <> "/") Then
            Throw New Exception(Oper)
        End If

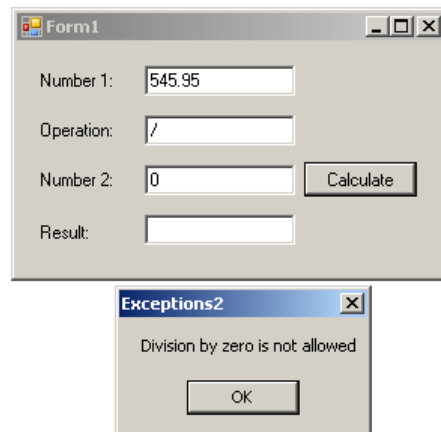
        If Oper = "/" Then
            If Operand2 = 0 Then
                Throw New DivideByZeroException(_
                    "Division by zero is not allowed")
            End If
        End If

        Result = Calculate(Operand1, Operand2, Oper)
        TextBox4.Text = CStr(Result)

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch ex As DivideByZeroException
    MsgBox(ex.Message)
Catch
    MsgBox("Invalid Operation: " & vbCrLf & _
        Oper & " is not a valid operator")
End Try
End Sub

```

This is an example of running the program:



You can still use regular functions that handle exceptions and each function can handle its own exception(s). Here is an example:

```

Private Function Addition(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Addition = Value1 + Value2
End Function

Private Function Subtraction(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Subtraction = Value1 - Value2
End Function

Private Function Multiplication(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Multiplication = Value1 * Value2
End Function

Private Function Division(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double

    Dim Result As Double = 0.0

    Try

        If Value2 = 0 Then
            Throw New DivideByZeroException("Division by zero is not allowed")
        End If

        Result = Value1 + Value2

    Catch ex As DivideByZeroException
        MsgBox(ex.Message)
    End Try

    Division = Result
End Function

```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."
Try
    Operand1 = Double.Parse(TextBox1.Text)
    Oper = TextBox2.Text
    Operand2 = Double.Parse(TextBox3.Text)
    If (Oper <> "+" ) And _
        (Oper <> "-" ) And _
        (Oper <> "*" ) And _
        (Oper <> "/" ) Then
        Throw New Exception(Oper)
    End If
    Select Case Oper
        Case "+"
            Result = Addition(Operand1, Operand2)
        Case "-"
            Result = Subtraction(Operand1, Operand2)
        Case "*"
            Result = Multiplication(Operand1, Operand2)
        Case "/"
            Result = Division(Operand1, Operand2)
    End Select
    TextBox4.Text = CStr(Result)
Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch ex As DivideByZeroException
    MsgBox(ex.Message)
Catch
    MsgBox("Invalid Operation: " & vbCrLf & Oper & " is not a valid operator")
End Try
End Sub
```

**Doing the balancing act
with your current salary?**

SAVINGS



EXPENSES

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



Techniques of Using Exceptions

Throwing an Exception

Ads by Google

As mentioned above, the **Exception class** is equipped with a **Message** property that carries a message for the error that occurred. We also mentioned that the message of this property may not be particularly useful to a user. Fortunately, you can create your own message and pass it to the **Exception** object. To be able to receive custom messages, the **Exception** class provides the following constructor:

```
Public Sub New(message As String)
```

Compare Excel tables

Powerful and handy add-on for Excel 2000-2007 files comparison.

www.office-excel.com

To use it, in the section where you are anticipating the error, type the **Throw** keyword followed by a **New** instance of the **Exception** class using the constructor that takes a string. Here is an example:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/") Then
            Throw New Exception(Oper)
        End If

        Select Case Oper
            Case "+"
                Result = Operand1 + Operand2

            Case "-"
                Result = Operand1 - Operand2

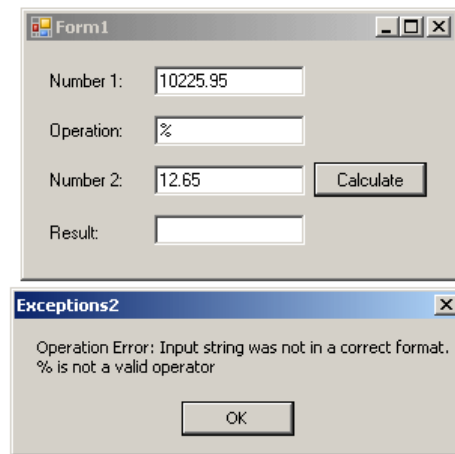
            Case "*"
                Result = Operand1 * Operand2

            Case "/"
                Result = Operand1 / Operand2

            Case Else
                MsgBox("Bad Operation")
        End Select

        TextBox4.Text = CStr(Result)

    Catch Ex As Exception
        MsgBox("Operation Error: " & Ex.Message & _
            vbCrLf & Oper & " is not a valid operator")
    End Try
End Sub
```



Catching Various Exceptions

In the above examples, when we anticipated some type of problem, we instructed the [compiler](#) to use our default catch section. We left it up to the compiler to find out when there was a problem and we provided a catch section to deal with it. A method with numerous or complex operations and requests can also produce different types of errors. With such a type of program, you should be able to face different problems and deal with them individually, each by its own kind. To do this, you can create different catch sections, each made for a particular error. The formula used would be:

```
Try
    ' Code to Try
Catch Arg1
    ' One Exception
Catch Arg2
    ' Another Exception
End Try
```

The compiler would proceed in a top-down:

1. Following the normal flow of the program, the compiler enters the **try** block
2. If no exception occurs in the **Try** block, the rest of the **Try** block is executed
If an exception occurs in the **Try** block, the compiler registers the type of error that occurred. If there is a **Throw** line, the compiler registers it also:
 - a. The compiler gets out of the **Try** section
 - b. The compiler examines the first **Catch**. If the first **Catch** matches the thrown error, that catch executes and the [exception handling](#) routine may seize. If the first **Catch** does not match the thrown error, the compiler proceeds with the next **Catch**
 - c. The compiler checks the next match, if any, and proceeds as in the first match. This continues until the compiler finds a **Catch** clause that matches the thrown error
 - d. If one of the catches matches the thrown error, its body executes. If no **Catch** matches the thrown error, the compiler calls the **Exception** class and uses the default message

Multiple catches are written if or when a try block is expected to throw different types of errors. For example, in our [calculator](#), we want to consider only the addition, the [subtraction](#), the multiplication, and the division. It is also likely that the user may type one or two invalid numbers. This leads us to know that our program can produce at least two types of errors at this time. Based on this, we can address them using two catch clauses as follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."
    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)
        If (Oper <> "+") And _
            (Oper <> "-") And _
```

```

        (Oper <> "*" ) And _
        (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If

    Select Case Oper

        Case "+"
            Result = Operand1 + Operand2

        Case "-"
            Result = Operand1 - Operand2

        Case "*"
            Result = Operand1 * Operand2

        Case "/"
            Result = Operand1 / Operand2

        Case Else
            MsgBox("Bad Operation")

    End Select

    TextBox4.Text = CStr(Result)

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch Ex As Exception
    MsgBox("Operation Error: " & Ex.Message & _
        vbCrLf & Oper & " is not a valid operator")
End Try
End Sub

```

This program works fine as long as the user types two valid numbers and a valid [arithmetic](#) operator. Anything else, such as an invalid number or an unexpected operator would cause an error to be thrown:

Obviously various bad things could happen when this program is running. Imagine that the user wants to perform a division. You need to tell the compiler what to do if the user enters the denominator as 0 (or 0.00). If this happens, one of the options you should consider is to display a message and get out. Fortunately, the [.NET Framework](#) provides the **DivideByZeroException** class to deal with an exception caused by division by zero. As done with the message passed to the **Exception** class, you can compose your own message and pass it to the **DivideByZeroException(string message)** constructor.

Exception is the parent of all exception classes. It corresponds to the type of Catch that takes no argument. Therefore, if you write various catch blocks, the one that either takes nor argument or is of the **Exception** type must be the last.

Here is an example that catches two types of exceptions:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
    Handles Button1.Click

    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/") Then
            Throw New Exception(Oper)
        End If

        Select Case Oper

            Case "+"
                Result = Operand1 + Operand2

            Case "-"
                Result = Operand1 - Operand2

            Case "*"
                Result = Operand1 * Operand2

            Case "/"
                If Operand2 = 0 Then
                    Throw New DivideByZeroException(_
                        "Division by zero is not allowed")
                End If

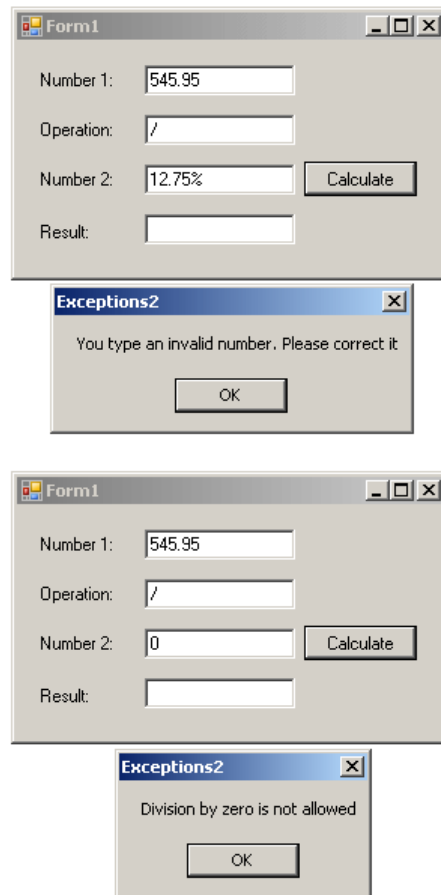
                Result = Operand1 / Operand2

            Case Else
                MsgBox("Bad Operation")

        End Select

        TextBox4.Text = CStr(Result)

    Catch ex As FormatException
        MsgBox("You type an invalid number. Please correct it")
    Catch ex As DivideByZeroException
        MsgBox(ex.Message)
    Catch
        MsgBox("Invalid Operation: " & vbCrLf & _
            Oper & " is not a valid operator")
    End Try
End Sub
```



❖ Practical Learning: Identifying the Thrown Exception

1. To catch various exceptions, change the code as follows:

```
Public Class Form1

    ' Order Information
    Dim CustomerName As String
    Dim OrderDate As DateTime
    Dim mm As String
    Dim dd As String
    Dim yyyy As String

    ' Quantities of items
    Dim NumberOfShirts As Integer
    Dim NumberOfPants As Integer
    Dim NumberOfOther As Integer

    ' Price of items
    Dim PriceOneShirt As Double
    Dim PriceAPairOfPants As Double
    Dim PriceOther As Double

    ' Each of these sub totals will be used for cleaning items
    Dim SubTotalShirts As Double
    Dim SubTotalPants As Double
    Dim SubTotalOther As Double

    ' Values used to process an order
    Dim TaxRate As Double
    Dim TotalOrder As Double
    Dim TaxAmount As Double
    Dim SalesTotal As Double

    Private Sub btnProcess_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) _
        Handles btnProcess.Click
        If btnProcess.Text = "Process" Then
            Height = 408
            btnProcess.Text = "Reset"
        Else
            Height = 232
            txtCustomerName1.Text = ""
        End If
    End Sub
End Class
```

```

txtMM.Text = "1"
txtDD.Text = "1"
txtYYYY.Text = "2000"
txtQtyShirts.Text = "0"
txtQtyPants.Text = "0"
txtQtyOther.Text = "0"
txtSubTotalShirts.Text = "0.00"
txtSubTotalPants.Text = "0.00"
txtSubTotalOther.Text = "0.00"

btnProcess.Text = "Process"
End If

' Request order information from the user
CustomerName = txtCustomerName1.Text

Try
    Dim mm As Integer, dd As Integer, yyyy As Integer
    mm = Integer.Parse(txtMM.Text)
    dd = Integer.Parse(txtDD.Text)
    yyyy = Integer.Parse(txtYYYY.Text)
    OrderDate = New DateTime(yyyy, mm, dd)

    ' This exception is thrown if the user types a value that cannot
    ' be converted into a valid number
Catch ex As FormatException
    MsgBox("Error: " & ex.Message & _
        vbCrLf & "The value you entered " & _
        "is not a valid number")

    ' This exception is thrown if the
    ' values that user had typed cannot
    ' produce a valid date value
Catch ex As ArgumentOutOfRangeException
    MsgBox("The date you entered is not valid" & _
        "- Please try again!")
End Try

' Request the quantity of each category of items
' Number of Shirts
Try
    NumberOfShirts = CInt(txtQtyShirts.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "shirts is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")

    ' This exception is thrown if the user types a negative value
Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Number of Pants
Try
    NumberOfPants = CInt(txtQtyPants.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "pair or pants is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")

Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Number of other items
Try
    NumberOfOther = CInt(txtQtyOther.Text)
Catch ex As FormatException
    MsgBox("The value you typed for the number of " & _
        "other items is not a valid number." & _
        vbCrLf & "Please enter a natural number such " & _
        "as 2 or 24 or even 248")

Catch ex As OverflowException
    MsgBox("The number you typed is negative but " & _
        "we cannot accept a negative number of shirts")
End Try

' Unit Prices of items
Try
    PriceOneShirt = CDbL(txtUnitPriceShirts.Text)
    If PriceOneShirt < 0 Then
        Throw New Exception("Negative numbers are not allowed " & _
            "for the price of a shirt")
    End If
End Try

```

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >



www.manashosting.com
Ads by Google

```

    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit price " & _
        "of a shirt is not a recognizable currency " & _
        "amount." & vbCrLf & _
        "Only natural or decimal numbers " & _
        "are allowed. Please consult the management " & _
        "to know the valid prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

Try
    PriceAPairOfPants = CDbI(txtUnitPricePants.Text)
    If PriceAPairOfPants < 0 Then
        Throw New Exception("Negative numbers are not allowed " & _
            "for the price of a pair of pants")
    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit price of " & _
        "a pair of pants is not a recognizable " & _
        "currency amount." & vbCrLf & _
        "Only natural or decimal " & _
        "numbers are allowed. You can consult the " & _
        "management to find out about " & _
        "the allowable prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

Try
    PriceOther = CDbI(txtUnitPriceOther.Text)
    If PriceOther < 0 Then
        Throw New Exception("Negative numbers are " & _
            "not allowed for the price")
    End If

Catch ex As FormatException
    MsgBox("The value you entered for the unit " & _
        "price of other items is not a valid amount." & _
        vbCrLf & "You must enter only a natural or a " & _
        "decimal number. For more information, " & _
        "please consult the management to get " & _
        "the right prices.")
Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

' Perform the necessary calculations
SubTotalShirts = NumberOfShirts * PriceOneShirt
SubTotalPants = NumberOfPants * PriceAPairOfPants
SubTotalOther = NumberOfOther * PriceOther

txtSubTotalShirts.Text = CStr(SubTotalShirts)
txtSubTotalPants.Text = CStr(SubTotalPants)
txtSubTotalOther.Text = CStr(SubTotalOther)

' Calculate the "temporary" total of the order
TotalOrder = SubTotalShirts + SubTotalPants + SubTotalOther

' Display the receipt
txtCustomerName2.Text = CustomerName
txtOrderDate.Text = FormatDateTime(OrderDate, DateFormat.LongDate)
txtTotalOrder.Text = CStr(TotalOrder)
End Sub

Private Sub btnTax_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnTax.Click

' Get the tax rate
Try
    TaxRate = CDbI(txtTaxRate.Text) / 100
    If TaxRate < 0 Then
        Throw New Exception("Negative numbers are not " & _
            "allowed for a tax rate")
    End If

Catch ex As FormatException
    MsgBox("The value you entered is not " & _
        "recognized as a valid tax rate." & _
        vbCrLf & "A valid tax rate is a value " & _
        "between 0 and 100.00" & _
        vbCrLf & "Please try again.")

```

```

Catch Ex As Exception
    MsgBox("Something bad happened")
End Try

' Calculate the tax amount using a constant rate
TaxAmount = TotalOrder * TaxRate
' Add the tax amount to the total order
SalesTotal = TotalOrder + TaxAmount

txtTaxAmount.Text = TaxAmount.ToString()
txtNetPrice.Text = SalesTotal.ToString()
End Sub

Private Sub btnDifference_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnDifference.Click

    Dim AmountTended As Double = 0.0
    Dim Difference As Double = 0.0

    ' Request money for the order
    Try
        AmountTended = CDb1(txtAmountTended.Text)
        If AmountTended < 0 Then
            Throw New Exception("Negative numbers are not " & _
                "allowed forthe amount tended")
        End If

    Catch ex As FormatException
        MsgBox("The value you entered for the amount " & _
            "tended is not valid. Only natural or " & _
            "decimal numbers are allowed." & _
            "Please try again.")
    Catch Ex As Exception
        MsgBox("Something bad happened")
    End Try

    ' Calculate the difference owed to the customer
    ' or that the customer still owes to the store
    Difference = AmountTended - SalesTotal

    txtDifference.Text = CStr(Difference)
End Sub
End Class

```

2. Test the [application](#) and return to your [programming environment](#)

Exceptions Nesting

The calculator simulator we have studied so far performs a division as one of its assignments. We learned that, in order to perform any operation, the compiler must first make sure that the user has entered a valid operator. Provided the operator is one of those we also must make sure that the user typed valid numbers. It is still possible that the user would enter 0 for the denominator, which depends on the exception that could result from a zero denominator depends on the user first entering a valid number for the denominator.

You can create an exception inside of another. This is referred to as nesting an exception. This is done by applying the same techniques used to nest conditional statements. This means that you can write an exception that depends on, and is subject to, another exception. To nest an exception, create a **Try** clause in the body of the parent exception. The nested **Try** clause must be followed by its own **Catch** clause(s). To effectively handle the exception, make sure you include an appropriate **Throw** in the **Try** block. Here is an example:

```

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _
            (Oper <> "-") And _
            (Oper <> "*") And _
            (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If
    End Try

```

```

Select Case Oper

    Case "+"
        Result = Operand1 + Operand2
        TextBox4.Text = CStr(Result)

    Case "-"
        Result = Operand1 - Operand2
        TextBox4.Text = CStr(Result)

    Case "*"
        Result = Operand1 * Operand2
        TextBox4.Text = CStr(Result)

    Case "/"
        Try
            If Operand2 = 0 Then
                Throw New DivideByZeroException(_
                    "Division by zero is not allowed")
            End If
            Result = Operand1 / Operand2

        Catch ex As DivideByZeroException
            MsgBox(ex.Message)
        End Try

    Case Else
        MsgBox("Bad Operation")

End Select

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch
    MsgBox("Invalid Operation: " & vbCrLf & _
        Oper & " is not a valid operator")
End Try
End Sub

```

Exceptions and Functions

One of the most effective techniques used to deal with code is to isolate assignments in different functions. For example, the **Select Case** statement that was performing the operations in the "normal" version of our program could be written as follows:

```

Private Function Calculate(ByVal Value1 As Double, _
    ByVal Value2 As Double, _
    ByVal symbol As Char) As Double

    Dim Result As Double = 0.0

    Select Case symbol

        Case "+"
            Result = Value1 + Value2

        Case "-"
            Result = Value1 - Value2

        Case "*"
            Result = Value1 * Value2

        Case "/"
            Result = Value1 / Value2
    End Select

    Calculate = Result
End Function

Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles Button1.Click

    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."

    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)

        If (Oper <> "+") And _

```

```

        (Oper <> "-") And _
        (Oper <> "**") And _
        (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If

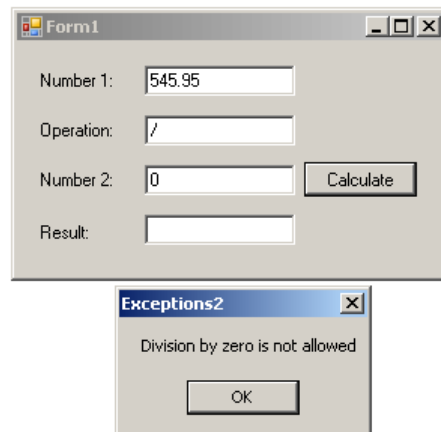
    If Oper = "/" Then
        If Operand2 = 0 Then
            Throw New DivideByZeroException(_
                "Division by zero is not allowed")
        End If
    End If

    Result = Calculate(Operand1, Operand2, Oper)
    TextBox4.Text = CStr(Result)

Catch ex As FormatException
    MsgBox("You type an invalid number. Please correct it")
Catch ex As DivideByZeroException
    MsgBox(ex.Message)
Catch
    MsgBox("Invalid Operation: " & vbCrLf & _
        Oper & " is not a valid operator")
End Try
End Sub

```

This is an example of running the program:



You can still use regular functions that handle exceptions and each function can handle its own exception(s). Here is an example:

```

Private Function Addition(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Addition = Value1 + Value2
End Function

Private Function Subtraction(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Subtraction = Value1 - Value2
End Function

Private Function Multiplication(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double
    Multiplication = Value1 * Value2
End Function

Private Function Division(ByVal Value1 As Double, _
    ByVal Value2 As Double) As Double

    Dim Result As Double = 0.0

    Try

        If Value2 = 0 Then
            Throw New DivideByZeroException("Division by zero is not allowed")
        End If

        Result = Value1 + Value2

    Catch ex As DivideByZeroException
        MsgBox(ex.Message)
    End Try

    Division = Result
End Function

```

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles Button1.Click
    Dim Operand1 As Double
    Dim Operand2 As Double
    Dim Result As Double = 0.0
    Dim Oper As String = "."
    Try
        Operand1 = Double.Parse(TextBox1.Text)
        Oper = TextBox2.Text
        Operand2 = Double.Parse(TextBox3.Text)
        If (Oper <> "+" ) And _
            (Oper <> "-" ) And _
            (Oper <> "*" ) And _
            (Oper <> "/" ) Then
            Throw New Exception(Oper)
        End If
        Select Case Oper
            Case "+"
                Result = Addition(Operand1, Operand2)
            Case "-"
                Result = Subtraction(Operand1, Operand2)
            Case "*"
                Result = Multiplication(Operand1, Operand2)
            Case "/"
                Result = Division(Operand1, Operand2)
        End Select
        TextBox4.Text = CStr(Result)
    Catch ex As FormatException
        MsgBox("You type an invalid number. Please correct it")
    Catch ex As DivideByZeroException
        MsgBox(ex.Message)
    Catch
        MsgBox("Invalid Operation: " & vbCrLf & Oper & " is not a valid operator")
    End Try
End Sub
```

A better paying job balances everything!					Register Now >>
 I.T. 11,400 jobs Search Now	 H.R. 10,956 jobs Search Now	 Finance 3,588 jobs Search Now	 B.P.O. 28,920 jobs Search Now	 Sales 10,956 jobs Search Now	TIMESJOBS.COM <i>Because you are worth more</i>

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



File Information

Introduction

In its high level of support for file processing, the [.NET Framework](#) provides the **FileInfo class**. This class is equipped to handle all types of file-related operations including creating, copying, [moving](#), renaming, or deleting a file. **FileInfo** is based on the **FileSystemInfo** class that provides information on characteristics of a file.

To assist you with finding information about a file, the **FileSystem** class from the **My** object is equipped with a method named **GetFileInfo**.

www.manashosting.com

Ads by Google

❖ Practical Learning: Introducing File Information

1. Start Microsoft [Visual Basic](#) and create a new Windows [Application](#) named **WattsALoan1**
2. In the Solution Explorer, right-click Form1.vb and click Rename
3. Type **WattsALoan.vb** and press Enter
4. Design the form as follows:

Control	Name	Text
Label		Acnt #:
Label		Customer Name:
Label		Customer:
TextBox	txtAccountNumber	
TextBox	txtCustomerName	
Label		Empl #:
Label		Employee Name:
Label		Prepared By:
TextBox	txtEmployeeNumber	
TextBox	txtEmployeeName	
Button	btnNewEmployee	
Label		Loan Amount:
TextBox	txtLoanAmount	TextAlign: Right
Label		Interest Rate:
TextBox	txtInterestRate	TextAlign: Right
Label		%

Label		Periods
TextBox		txtPeriods TextAlign: Right
Button	btnCalculate	Calculate
Label		Monthly Payment:
TextBox	txtMonthlyPayment	
Button	btnClose	Close

- Right-click the form and click View Code
- Just above the Public Class line, import the [System.IO](#) namespace:

```
Imports System.IO
```

```
Public Class WattsALoan
```

- In the Class Name combo box, select btnCalculate
- In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
    Handles btnCalculate.Click

    Dim LoanAmount As Double
    Dim InterestRate As Double
    Dim Periods As Double
    Dim MonthlyPayment As Double

    Try
        LoanAmount = CDb1(txtLoanAmount.Text)
    Catch ex As Exception
        MsgBox("Invalid Loan Amount")
    End Try

    Try
        InterestRate = CDb1(txtInterestRate.Text)
    Catch ex As Exception
        MsgBox("Invalid Interest Rate")
    End Try

    Try
        Periods = CDb1(txtPeriods.Text)
    Catch ex As Exception
        MsgBox("Invalid Periods Value")
    End Try

    MonthlyPayment = Pmt(InterestRate / 12 / 100, _
                        Periods, -LoanAmount, 0, _
                        DueDate.BegOfPeriod)
    txtMonthlyPayment.Text = FormatCurrency(MonthlyPayment)
End Sub
```

- In the Class Name combo box, select btnClose
- In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnClose_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
    Handles btnClose.Click

    End
End Sub
```

- Save the file

File Initialization

The **FileInfo** class is equipped with one constructor whose syntax is:

```
Public Sub New(fileName As String)
```

This constructor takes as argument the name of a file or its complete path. If you provide only the name of the file, the [compiler](#) would consider the same directory of its project.

As mentioned previously, to [get information](#) about a file, you can call the **GetFileInfo()** method of the **FileSystem** class from the **My** object. Its syntax is:

```
Public Shared Function GetFileInfo(file As String) As FileInfo
```

This shared [method returns](#) a **FileInfo** object. Here is an example of calling it:

```
Imports System.IO

Public Class Exercise

    Private Filename As String

    Private Sub btnFileInformation_Click(ByVal sender As System.Object, _
                                        ByVal e As System.EventArgs) _
        Handles btnFileInformation.Click

        Dim PeopleInformation As FileInfo

        PeopleInformation = My.Computer.FileSystem.GetFileInfo(Filename)
    End Sub
End Class
```

After calling this method, you can then use its returned value to get the information you want about the file.

❖ Practical Learning: Initializing a File

1. In the Class Name combo box, select (WattsALoan Events)
2. In the Method Name combo box, select Load and implement the event as follows:

```
Private Sub WattsALoan_Load(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
    Handles Me.Load

    Dim Filename As String = "Employees.wal"
    Dim EmployeeInformation As FileInfo = _
        My.Computer.FileSystem.GetFileInfo(Filename)

End Sub
```

3. Save the file

File Creation

The **FileInfo** constructor is mostly meant only to indicate that you want to use a file, whether it exists already or it would be created. Based on this, if you execute an application that has only a **FileInfo** [object created](#) using the constructor as done above, nothing would happen.

To create a file, you have various alternatives. If you want to create one without [writing](#) anything in it, which implies creating an empty file, you can call the **FileInfo.Create()** method. Its syntax is:

```
Public Function Create As FileStream
```

This method simply creates an empty file. Here is an example of calling it:

```
Private Sub btnFileInformation_Click(ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
    Handles btnFileInformation.Click

    Dim PeopleInfo As FileInfo = New FileInfo("People.txt")
    PeopleInfo.Create()

End Sub
```

The **FileInfo.Create()** method returns a **FileStream** object. You can use this returned value to write any type of value into the file, including text. If you want to create a file that contains text, an alternative is to call the **FileInfo.CreateText()** method. Its syntax is:

```
Public Function CreateText As StreamWriter
```

This method returns a **StreamWriter** object. You can use this returned object to write text to the file.

File Existence

When you call the **FileInfo.Create()** or the **FileInfo.CreateText()** method, if the file passed as argument, or as the file in the path of the argument, exists already, it would be deleted and a new one would be created with the same name. This can cause an important file to be deleted. Therefore, before creating a file, you may need to check whether it exists already. To do this, you can check the value of the Boolean **FileInfo.Exists** property. This property holds a **True** value if the file exists already and it holds a **False** value if the file does not yet exist or it does not exist in the path. Here is an example of calling it:

```
Private Sub btnFileInformation_Click(ByVal sender As System.Object, _
                                    ByVal e As System.EventArgs) _
```

```

                                Handles btnFileInformation.Click
Dim Filename As String
Dim PeopleInformation As FileInfo

Filename = "Student12.std"
PeopleInformation = My.Computer.FileSystem.GetFileInfo(Filename)

If PeopleInformation.Exists = True Then
    MsgBox("The file exists already")
Else
    MsgBox("Unknown file")
End If
End Sub

```

❖ Practical Learning: Creating a Text File

1. Change the Load event of the form as follows:

```

Private Sub WattsALoan_Load(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim EmployeeWriter As StreamWriter
    Dim Filename As String = "Employees.wal"
    Dim EmployeeInformation As FileInfo = _
        My.Computer.FileSystem.GetFileInfo(Filename)

    ' If the employees file was not created already,
    ' then create it
    If Not EmployeeInformation.Exists Then
        EmployeeWriter = EmployeeInformation.CreateText()
    End If
End Sub

```

2. Save the file

Writing to a File

As mentioned earlier, the **FileInfo.Create()** and the **FileInfo.CreateText()** methods can be used to create a file but they not write values to the file. To write values in the file, each method returns an appropriate object. The **FileInfo.Create()** method returns **FileStream** object. You can use this to specify the type of operation that would be allowed on the file. To write normal text to a file, you can first call the **FileInfo.CreateText()** method that returns a **StreamWriter** object. Here is an example:

```

Private Sub btnSave_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles btnSave.Click
    Dim Filename As String
    Dim StudentsWriter As StreamWriter
    Dim StudentInformation As FileInfo

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)
    StudentsWriter = StudentInformation.CreateText()

    Try
        StudentsWriter.WriteLine(txtFirstName.Text)
        StudentsWriter.WriteLine(txtLastName.Text)
        StudentsWriter.WriteLine(cbxGenders.SelectedIndex)

        txtFirstName.Text = ""
        txtLastName.Text = ""
        cbxGenders.SelectedIndex = 2
    Finally
        StudentsWriter.Close()
    End Try
End Sub

```

As an alternative to **Create()** or **CreateText()**, if you want to create a file that can only be written to, you can call the **FileInfo.OpenWrite()** method. Its syntax is:

```
Public Function OpenWrite As FileStream
```

This method returns a **FileStream** that you can then use to write values into the file.

❖ Practical Learning: Writing to a Text File

1. Change the Load event of the form as follows:

```

Private Sub WattsALoan_Load(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles Me.Load

    Dim EmployeeWriter As StreamWriter
    Dim Filename As String = "Employees.wal"
    Dim EmployeeInformation As FileInfo = _
        My.Computer.FileSystem.GetFileInfo(Filename)

    ' If the employees file was not created already,
    ' then create it
    If Not EmployeeInformation.Exists Then
        EmployeeWriter = EmployeeInformation.CreateText()

        ' And create a John Doe employee
        Try
            EmployeeWriter.WriteLine("00-000")
            EmployeeWriter.WriteLine("John Doe")
        Finally
            EmployeeWriter.Close()
        End Try
    End If
End Sub

```

2. Save the file

Appending to a File

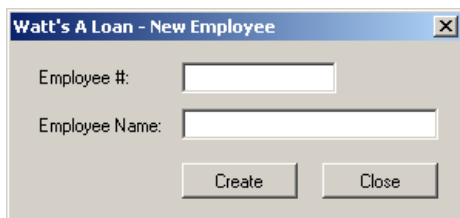
You may have created a text-based file and written to it. If you open such a file and find out that a piece of information is missing, you can add that information to the end of the file. To do this, you can call the **FileInfo.AppendText()** method. Its syntax is:

```
Public Function AppendText As StreamWriter
```

When calling this method, you can retrieve the **StreamWriter** object that it returns, then use that object to add new information to the file.

❖ Practical Learning: Writing to a Text File

1. To create a new form, on the main menu, click Project -> Add Windows Form...
2. In the Templates list, make sure Windows Form is selected. Set the Name to **NewEmployee** and click Add
3. Design the form as follows:



Control	Text	Name
Label	Employee #:	
TextBox		txtEmployeeNumber
Label	Employee Name:	
TextBox		txtEmployeeName
Button	Create	btnCreate
Button	Close	btnClose

4. Right-click the form and click View Code
5. In the top section of the file, just above the Public Class line, import the **System.IO** namespace:

```
Imports System.IO
```

```
Public Class NewEmployee
```

```
End Class
```

6. In the Class Name combo box, select btnCreate
7. In the Method Name combo box, select Click and implement the event as follows:

Ads by Google



[Bulk SMS - Stock Brokers](#)

Stock Alert Mangement system -Best Prices - Pay only 4 delivered SMS
www.Planet41.com

[Free Computer eBooks](#)

10,000+ Online Computer Books. all are free!
2020ok.com

[C/C++ Programmers needed](#)

Join GetAFreelancer.com and bid on projects. Free and quick signup.
www.GetAFreelancer.com

[Text to Speech Voices](#)

Astounding text to speech software! Download free trial.
www.nextup.com

[Jokes, Funny Pics & Video](#)

Share & Read Jokes, Funny Videos & Pics With Your Friends On Minglebox
www.minglebox.com

```

Private Sub btnCreate_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles btnCreate.Click
    Dim Filename As String = "Employees.wal"
    Dim EmployeeInformation As FileInfo = New FileInfo(Filename)
    Dim EmployeeWriter As StreamWriter

    ' Normally, we should have the file already but just in case...
    If Not EmployeeInformation.Exists Then
        EmployeeWriter = EmployeeInformation.CreateText()
    Else ' If the file exists already, then we will only add to it
        EmployeeWriter = EmployeeInformation.AppendText()
    End If

    Try
        EmployeeWriter.WriteLine(txtEmployeeNumber.Text)
        EmployeeWriter.WriteLine(txtEmployeeName.Text)
    Finally
        EmployeeWriter.Close()
    End Try

    txtEmployeeNumber.Text = ""
    txtEmployeeName.Text = ""
    txtEmployeeNumber.Focus()
End Sub

```

8. In the Class Name combo box, select btnClose
9. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub btnClose_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles btnClose.Click
    Close()
End Sub

```

10. In the Solution Explorer, right-click WattsALoan.vb and click View Code
11. In the Class Name combo box, select btnNewEmployee
12. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub btnNewEmployee_Click(ByVal sender As Object, _
                                  ByVal e As System.EventArgs) _
                                  Handles btnNewEmployee.Click
    Dim FormEmployee As NewEmployee = New NewEmployee()

    FormEmployee.ShowDialog()
End Sub

```

13. In the Class Name combo box, select txtEmployeeNumber
14. In the Method Name combo box, select Leave
15. Implement the event as follows:

```

Private Sub txtEmployeeNumber_Leave(ByVal sender As Object, _
                                    ByVal e As System.EventArgs) _
                                    Handles txtEmployeeNumber.Leave
    Dim Found As Boolean
    Dim Filename As String
    Dim EmployeeReader As StreamReader
    Dim EmployeeInformation As FileInfo
    Dim EmployeeNumber As String, EmployeeName As String

    Filename = "Employees.wal"
    EmployeeInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    If EmployeeInformation.Exists Then
        If txtEmployeeNumber.Text = "" Then
            txtEmployeeName.Text = ""
            Exit Sub
        Else
            EmployeeReader = EmployeeInformation.OpenText()

            Try
                Do
                    EmployeeNumber = EmployeeReader.ReadLine()

                    If EmployeeNumber = txtEmployeeNumber.Text Then
                        EmployeeName = EmployeeReader.ReadLine()
                        txtEmployeeName.Text = EmployeeName
                    End If
                Loop While EmployeeReader.ReadLine() IsNot Nothing
            Finally
                EmployeeReader.Close()
            End Try
        End If
    End If
End Sub

```

```

        Found = True
    End If
    Loop While EmployeeReader.Peek() >= 0

    ' When the application has finished checking the file
    ' if there was no employee with that number, let the user know
    If Found = False Then

        MsgBox("No employee with that number was found")
        txtEmployeeName.Text = ""
        txtEmployeeNumber.Focus()
    End If
    Finally
        EmployeeReader.Close()
    End Try
End If
End If
End Sub

```

16. Execute the application to test it
17. First create a few employees as follows:

Employee #	Employee Name
42-806	Patricia Katts
75-148	Helene Mukoko
36-222	Frank Leandro
42-808	Gertrude Monay

18. Process a [loan](#)

19. Close the application

Reading from a File

As opposed to writing to a file, you can read from it. To support this, the **FileInfo** class is equipped with a method named **OpenText()**. Its syntax is:

```
Public Function OpenText As StreamReader
```

This method returns a **StreamReader** object. You can then use this object to read the lines of a text file. Here is an example:

```

Private Sub btnOpen_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnOpen.Click

    Dim Filename As String
    Dim StudentsReader As StreamReader
    Dim StudentInformation As FileInfo

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)
    StudentsReader = StudentInformation.OpenText

    Try
        txtFirstName.Text = StudentsReader.ReadLine
        txtLastName.Text = StudentsReader.ReadLine
        cbxGenders.SelectedIndex = CInt(StudentsReader.ReadLine)

    Finally
        StudentsReader.Close()
    End Try
End Sub

```

If you want to open a file that can only be read from, you can call the **FileInfo.OpenRead()** method. Its syntax is:

```
Public Function OpenRead As FileStream
```

This method returns a **FileStream** that you can then use to read values from the file.

Routine Operations on Files

Opening a File

As opposed to creating a file, probably the second most regular operation performed on a file consists of opening it to read or explore its contents. To support opening a file, the **FileInfo** class is equipped with the **Open()** method that is overloaded with three versions. Their syntaxes are:

```
Public Function Open ( _
    mode As FileMode _
) As FileStream
Public Function Open ( _
    mode As FileMode, _
    access As FileAccess _
) As FileStream
Public Function Open ( _
    mode As FileMode, _
    access As FileAccess, _
    share As FileShare _
) As FileStream
```

You can select one of these methods, depending on how you want to open the file, using the options for **file mode**, **file access**, and **file sharing**. Each version of this method returns a **FileStream** object that you can then use to process the file. After opening the file, you can then read or use its content.

Deleting a File

If you have an existing file you don't need anymore, you can delete it. This operation can be performed by calling the **FileInfo.Delete()** method. Its syntax is:

```
Public Overrides Sub Delete
```

Here is an example:

```
Private Sub btnDelete_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnDelete.Click
    Dim Filename As String
    Dim StudentInformation As FileInfo

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    If PeopleInformation.Exists = True Then
        StudentInformation.Delete()
    Else
        MsgBox("Unknown file")
    End If
End Sub
```

Copying a File

You can make a copy of a file from one directory to another. To do this, you can call the **FileInfo.CopyTo()** method that is overloaded with two versions. One of the versions has the following syntax:

```
public FileInfo CopyTo(string destFileName)
```

When calling this method, specify the path or directory that will be the [destination](#) of the copied file. Here is an example:

```
Private Sub btnCopy_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnCopy.Click
    Dim Filename As String
    Dim StudentInformation As FileInfo
    Dim MyDocuments As String = _
        Environment.GetFolderPath(Environment.SpecialFolder.Personal)
```



```

Filename = "Student1.std"
StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)

If StudentInformation.Exists = True Then
    StudentInformation.CopyTo(MyDocuments & "\Federal.txt")
Else
    MsgBox("Unknown file")
End If
End Sub

```

In this example, a file named Reality.txt in the directory of the project would be retrieved and its content would be applied to a new file named Federal.txt created in the My [Documents folder](#) of the current user.

When calling the first version of the **FileInfo.CopyTo()** method, if the file exists already, the operation would not continue and you would simply receive a message box. If you insist, you can overwrite the target file. To do this, you can use the second version of this method. Its syntax is:

```
Public Function CopyTo(destFileName As String, overwrite As Boolean) As FileInfo
```

The first argument is the same as that of the first version of the method. The second argument specifies what action to take if the file exists already in the target directory. If you want to overwrite it, pass the second argument as true; otherwise, pass it as false. Here is an example:

```

Private Sub btnCopy_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnCopy.Click

    Dim Filename As String
    Dim StudentInformation As FileInfo
    Dim MyDocuments As String = _
        Environment.GetFolderPath(Environment.SpecialFolder.Personal)

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    If StudentInformation.Exists = True Then
        StudentInformation.CopyTo(MyDocuments & "\Federal.txt", True)
    Else
        MsgBox("Unknown file")
    End If
End Sub

```

Moving a File

If you copy a file from one directory to another, you would have two copies of the same file or the same contents in two files. Instead of copying, if you want, you can simply move a file from one directory to another. This operation can be performed by calling the **FileInfo.MoveTo()** method. Its syntax is:

```
Public Sub MoveTo(destFileName As String)
```

The argument to this method is the same as that of the **CopyTo()** method. After executing this method, the **FileInfo** object would be moved to the *destFileName* path.

Here is an example:

```

Private Sub btnMove_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnMove.Click

    Dim Filename As String
    Dim StudentInformation As FileInfo
    Dim MyDocuments As String = _
        Environment.GetFolderPath(Environment.SpecialFolder.Personal)

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    If StudentInformation.Exists = True Then
        StudentInformation.MoveTo(MyDocuments & "\Federal.txt")
    Else
        MsgBox("Unknown file")
    End If
End Sub

```

Unlimited Webspace and Unlimited Bandwidth **FREE** **Only**
Domain Registration **Rs. 2000/-**

- ⇒ 100 Email Ids of 10 GB
- ⇒ All Database FREE
- ⇒ 5000 Templates / Website Builder
- ⇒ Full Control Panel
- ⇒ 200 SEO Tools/Softwares/Weblinks

www.manashosting.com Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.



File System Information

The Date and Time a File Was Created

After a file has been created, the [operating system](#) makes a note of the date and the time the file was created. This information can be valuable in other operations such as search routines. You too are allowed to change this date and time values to those you prefer.

As mentioned already, the OS makes sure to keep track of the date and time a file was created. To find out what those date and time values are, you can access the get accessor of the **FileSystemInfo.CreationTime** property, which is of type **DateTime**. Here is an example of using it:

```
Private Sub btnCreationDate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnCreationDate.Click

    Dim Filename As String
    Dim FileCreationTime As DateTime
    Dim StudentInformation As FileInfo

    Filename = "Student1.std"
    StudentInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    If StudentInformation.Exists = True Then
        FileCreationTime = StudentInformation.CreationTime
        MsgBox(FormatDateTime(FileCreationTime, DateFormat.LongDate))
    Else
        MsgBox("Unknown file")
    End If

End Sub
```

Of course, by formatting the value, you can get only either the date or only the time.

If you don't like the date, the time, or both, that the OS would have set when the file was created, you can change them. To change one or both of these values, you can assign a desired **DateTime** object to the set accessor of the **FileSystemInfo.CreationTime** property.

The Date and Time a File Was Last Accessed

Many [applications](#) allow a user to open an existing file and to modify it. When people work in a team or when a particular file is regularly opened, at one particular time, you may want to know the date and time that the file was last accessed. To get this information, you can access the **FileSystemInfo.LastAccessTime** property, which is of type **DateTime**.

If you are interested to know the last date and time a file was modified, you can get the value of its **FileSystemInfo.LastWriteTime** property, which is of type **DateTime**.

The Name of a File

The operating system requires that each file have a name. In fact, the name must be specified when creating a file. This allows the OS to catalogue the computer files. This also allows you to locate or identify a particular file you need.

When reviewing or opening a file, to get its name, the **FileInfo** class is equipped with the **Name** property. Here is an example:

```
MsgBox("The name of this file is: \" + fileLoan.Name & "\"")
```

This string simply identifies a file.

The Extension of a File

With the advent of [Windows 95](#) and later, the user doesn't have to specify the extension of a

file when creating it. Because of the type of confusion that this can lead to, most applications assist the user with this detail. Some applications allow the user to choose among various extensions. For example, using [Notepad](#), a user can open a text, a PHP, a script, or an HTML file.

When you access a file or when the user opens one, to know the extension of the file, you can access the value of the **FileSystemInfo.Extension** property. Here is an example:

```
MsgBox("File Extension: " & fileLoan.Extension)
```

The Size of a File

One of the routine operations the operating system performs consists of calculating the size of files it holds. This information is provided in terms of bits, kilobits, or kilobytes. To get the size of a file, the **FileInfo** class is quipped with the **Length** property. Here is an example of accessing it:

```
MsgBox("File Size: " & fileLoan.Length.ToString())
```

The Path to a File

Besides its name, a file must be located somewhere. The location of a file is referred to as its path or directory. The **FileInfo** class represents this path as the **DirectoryName** property. Therefore, if a file has already been created, to get its path, you can access the value of the **FileInfo.DirectoryName** property.

Besides the **FileInfo.Directoryname**, to know the full path to a file, you can access its **FileSystemInfo.FullName** property.

The Attributes of a File

Attributes are characteristics that apply to a file, defining what can be done or must be disallowed on it. The Attributes are primarily defined by, and in, the operating system, mostly when a file is created. When the user accesses or opens a file, to get its attributes, you can access the value of its **FileSystemInfo.Attributes** property. This property produces a **FileAttributes** object.

When you create or access a file, you can specify or change some of the attributes. To do this, you can create a **FileAttributes** object and assign it to the **FileSystemInfo.Attributes** property.

FileAttributes is an enumeration with the following members: **Archive**, **Compressed**, **Device**, **Directory**, **Encrypted**, **Hidden**, **Normal**, **NotContentIndexed**, **Offline**, **ReadOnly**, **ReparsePoint**, **SparseFile**, **System**, and **Temporary**.



Coding

genius?

Prove

it.



Coding genius? Prove it.



[Home](#)

Copyright © 2008 FunctionX, Inc.

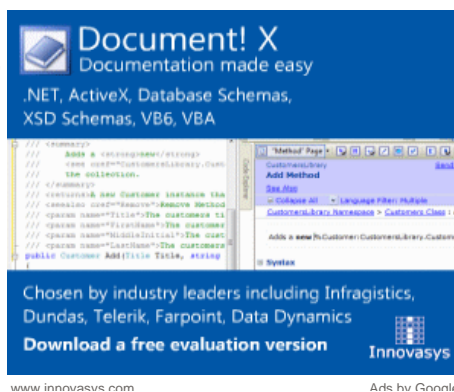


Directories

Introduction

A directory is a section of a medium (floppy disc, flash drive, hard drive, CD, DVD, etc) used to delimit a group of files. Because it is a "physical" area, it can handle operations not available on files. In fact, there are many fundamental differences between both:

- A file is used to contain data. A directory doesn't contain data
- A directory can contain one or more files and not vice-versa
- A directory can contain other directories
- A file can be moved from one directory to another. This operation is not possible vice-versa since a file cannot contain a directory



The similarities of both types are:

- A directory or a file can be created. One of the restrictions is that two files cannot have the same name inside of the same directory. Two directories cannot have the same name inside of the same parent directory.
- A directory or a file can be renamed. If a directory is renamed, the "path" of its file(s) changes
- A directory or a file can be deleted. If a directory is deleted, its files are deleted also
- A directory or a file can be moved. If a directory moves, it "carries" all of its files to the new location
- A directory or a file can be copied. A file can be copied from one directory to another. If a directory is copied to a new location, all of its files are also copied to the new location

❖ Practical Learning: Introducing Directories

1. Create a new Windows Application, named **WattsALoan2**
2. In the Solution Explorer, right-click Form1.vb and click Rename
3. Type **WattsALoan.vb** and press Enter
4. Design the form as follows:

Control	Name	Text
Label		If this is a new loan, enter a new account number and

		the name of the customer who is requesting the loan
Label		To open a previously prepared loan, enter its account number and press Tab
Label		Acnt #:
Label		Customer Name:
Label		Customer:
TextBox	txtAccountNumber	
TextBox	txtCustomerName	
Label		Empl #:
Label		Employee Name:
Label		Prepared By:
TextBox	txtEmployeeNumber	
TextBox	txtEmployeeName	
Button	btnNewEmployee	
Button	btnNewCustomer	
Label		Loan Amount:
TextBox	txtLoanAmount	
Label		Interest Rate:
TextBox	txtInterestRate	
Label		%
Label		Periods
TextBox		txtPeriods
Button	btnCalculate	Calculate
Label		Monthly Payment:
TextBox	txtMonthlyPayment	
Button	btnClose	Close

5. Double-click the Calculate button and implement its event as follows:

```
Imports System.IO

Public Class WattsALoan

    Private Sub btnCalculate_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
        Handles btnCalculate.Click

        Dim LoanAmount As Double
        Dim InterestRate As Double
        Dim Periods As Double
        Dim MonthlyPayment As Double

        Try
            LoanAmount = CDb1(txtLoanAmount.Text)
        Catch ex As Exception
            MsgBox("Invalid Loan Amount")
        End Try

        Try
            InterestRate = CDb1(txtInterestRate.Text)
        Catch ex As Exception

            MsgBox("Invalid Interest Rate")
        End Try

        Try
            Periods = CDb1(txtPeriods.Text)
        Catch ex As Exception
            MsgBox("Invalid Periods Value")
        End Try

        MonthlyPayment = Pmt(InterestRate / 12 / 100, _
                             Periods, -LoanAmount, 0, _
                             DueDate.BegOfPeriod)
        txtMonthlyPayment.Text = FormatCurrency(MonthlyPayment)
    End Sub
End Class
```

6. In the Class Name combo box, select btnClose
7. In the Method Name combo box, select Click and implement the event as follows:

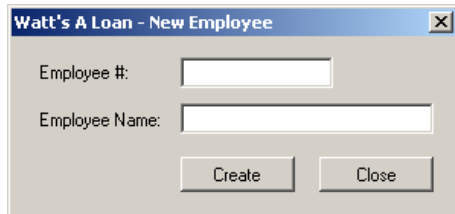
```
Private Sub btnClose_Click(ByVal sender As Object, _
```

```

        ByVal e As System.EventArgs) _
        Handles btnClose.Click
    End Sub
End Sub

```

8. To create a new form, on the main menu, click Project -> Add Windows Form...
9. In the Templates list, make sure Windows Form is selected. Set the Name to **NewEmployee** and click Add
10. Design the form as follows:



Control	Text	Name
Label	Employee #:	
TextBox		txtEmployeeNumber
Label	Employee Name:	
TextBox		txtEmployeeName
Button	Create	btnCreate
Button	Close	btnClose

11. Double-click the Close button
12. Implement the event as follows:

```

Imports System.IO

Public Class NewEmployee

    Private Sub btnClose_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles btnClose.Click

        Close()
    End Sub
End Class

```

13. In the Solution Explorer, right-click WattsALoan.vb and click View Code
14. In the Class Name combo box, select btnNewEmployee
15. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub btnNewEmployee_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnNewEmployee.Click

    Dim FormEmployee As NewEmployee = New NewEmployee()

    FormEmployee.ShowDialog()
End Sub

```

16. Save the file

Directory Creation

Before using a directory, you must first have it. You can use an existing directory if the operating system or someone else had already created one. You can also create a new directory. Directories are created and managed by various classes but the fundamental class is called **Directory**. **Directory** is a static class. All of its methods are static, which means you will never need to declare an instance of the **Directory** class in order to use it.

Besides the **Directory** class, additional operations of folders and sub-folders can be performed using the **DirectoryInfo** class.

To create a directory, you can call the **CreateDirectory()** method of the **Directory** class. This method is available in two versions. One of the versions uses the following syntax:

```
Public Shared Function CreateDirectory(path As String) As DirectoryInfo
```

This method takes as argument the (complete) path of the desired directory. Here is an example:

```
E:\Programs\Business Orders\Customer Information
```

When this method is called:

1. It first checks the parent drive, in this case E.
If the drive doesn't exist, because this method cannot create a drive, the compiler would throw a **DirectoryNotFoundException** exception
2. If the drive (in this case E) exists, the compiler moves to the first directory part of the path; in this case this would be the Programs folder in the E drive.
If the folder doesn't exist, the compiler would create it. If that first director doesn't exist, this means that the other directory(ies), if any, under the first don't exist. So, the compiler would create it/them
3. If the first directory exists and if there is no other directory under that directory, the compiler would stop and would not do anything further.
4. If the directory exists and there is a sub-directory specified under it, the compiler would check the existence of that directory.
If the sub-directory exists, the compiler would not do anything further and would stop.
If the sub-directory doesn't exist, the compiler would create it
5. The compiler would repeat step 4 until the end of the specified path

The **Directory.CreateDirectory()** method returns a **DirectoryInfo** object that you can use as you see fit.

Ads by Google



❖ Practical Learning: Creating a Directory

1. In the Class Name combo box, select (WattsALoan Events)
2. In the Method Name combo box, select Load and implement the event as follows:

```
Private Sub WattsALoan_Load(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
    Handles Me.Load
    Dim Folder As String
    Dim EmployeeWriter As StreamWriter
    Dim Filename As String = "Employees.wal"
    Dim EmployeeInformation As FileInfo = _
        My.Computer.FileSystem.GetFileInfo(Filename)

    Folder = "C:\Watts A Loan"

    If Not Directory.Exists(Folder) Then
        Directory.CreateDirectory(Folder)

        Dim strFilename As String = Folder & "\Employees.wal"

        Dim fiEmployees As FileInfo = New FileInfo(strFilename)

        ' If the employees file was not created already,
        ' then create it
        If Not EmployeeInformation.Exists Then
            EmployeeWriter = EmployeeInformation.CreateText()

            ' And create a John Doe employee
            Try
                EmployeeWriter.WriteLine("00-000")
                EmployeeWriter.WriteLine("John Doe")
            Finally
                EmployeeWriter.Close()
            End Try
        End If
    End If
End Sub
```

3. In the Solution Explorer, right-click NewEmployee and click View Code
4. In the Class Name combo box, select btnCreate
5. In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnCreate_Click(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
    Handles btnCreate.Click
    Dim Filename As String = "C:\Watts A Loan\Employees.wal"
    Dim EmployeeInformation As FileInfo
    Dim EmployeeWriter As StreamWriter

    EmployeeInformation = My.Computer.FileSystem.GetFileInfo(Filename)

    ' Normally, we should have the file already but just in case...
    If Not EmployeeInformation.Exists Then
        EmployeeWriter = EmployeeInformation.CreateText()
    Else ' If the file exists already, then we will only add to it
```

C# examples

C, C++, and C# Resources. Find tutorials, tips, and reviews.
www.DevSource.com

Microsoft .Net Training

Learn VB, ASP, ABP .net, XML Register now for summer in Delhi
www.appinonline.com

Free Datagrid for WPF

100% stylable and templatable, with rich in-place editing & more
xceed.com/Grid_WPF_Intro.html

Watch files and folders

Run scripts when files or folders change. Download free trial today.
www.torgesta.com/mytrigger/

Visual Basic Code Library

Open Source Code Snippet Library. Free Community for Developers.
www.daniweb.com/code


```

        EmployeeWriter = EmployeeInformation.AppendText()
    End If

    Try
        EmployeeWriter.WriteLine(txtEmployeeNumber.Text)
        EmployeeWriter.WriteLine(txtEmployeeName.Text)
    Finally
        EmployeeWriter.Close()
    End Try

    txtEmployeeNumber.Text = ""
    txtEmployeeName.Text = ""
    txtEmployeeNumber.Focus()
End Sub

```

- Click the WattsALoan.vb [Design] tab

Checking for a Directory Existence

Before using or creating a directory, you can first check if it exists. This is because, if a directory already exists in the location where you want to create it, you would be prevented from creating one with the same name. In the same way, if you just decide to directly use a directory that doesn't exist, the operation you want to perform may fail because the directory would not be found.

To check whether a directory exists or not, you can call the **Directory.Exists()** Boolean static method. Its syntax is:

```
Public Shared Function Exists(path As String) As Boolean
```

This method receives the (complete) path of the directory. If the path exists, the method returns true. If the directory doesn't exist, the method returns false.

Locating a File

One of the most routine operations performed in a directory consists of looking for a file. Microsoft Windows operating systems and the user's intuition have different ways of addressing this issue. The .NET Framework also provides its own means of performing this operation, through various techniques. You can start by checking the sub-directories and files inside of a main directory.

To look for files in a directory, the **DirectoryInfo** class can assist you with its **GetFiles()** method, which is overloaded with three versions.

❖ Practical Learning: Using Directories and Files

- In the Class Name combo box, select txtAccountNumber
- In the Method Name combo box, select Leave and implement the event as follows:

```

Private Sub txtAccountNumber_Leave(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
    Handles txtAccountNumber.Leave

    Dim Filename As String
    Dim LoanPath As String
    Dim FileFullname As String
    Dim ListOfLoans() As FileInfo
    Dim LoanReader As StreamReader
    Dim LoanFolder As DirectoryInfo
    Dim LoanInformation As FileInfo
    Dim Found As Boolean

    Found = False
    LoanPath = "C:\Watts A Loan"

    LoanFolder = New DirectoryInfo(LoanPath)
    ListOfLoans = LoanFolder.GetFiles("*", _
        SearchOption.AllDirectories)
    Filename = txtAccountNumber.Text & ".wal"
    FileFullname = LoanPath & "none.wal"

    For Each file As FileInfo In ListOfLoans
        If file.Name = Filename Then
            Found = True
            FileFullname = file.FullName
        End If
    Next

    If Found = True Then

        LoanInformation = My.Computer.FileSystem.GetFileInfo(Filename)
        LoanReader = LoanInformation.OpenText
    End If
End Sub

```

```

    Try
        txtAccountNumber.Text = LoanReader.ReadLine
        txtCustomerName.Text = LoanReader.ReadLine
        txtEmployeeNumber.Text = LoanReader.ReadLine
        txtEmployeeName.Text = LoanReader.ReadLine
        txtLoanAmount.Text = LoanReader.ReadLine
        txtInterestRate.Text = LoanReader.ReadLine
        txtPeriods.Text = LoanReader.ReadLine
        txtMonthlyPayment.Text = LoanReader.ReadLine
    Finally
        LoanReader.Close()
    End Try
End If
End Sub

```

3. In the Class Name combo box, select txtEmployeeNumber

4. In the Method Name combo box, select click Leave and implement the event as follows:

```

Private Sub txtAccountNumber_Leave(ByVal sender As Object, _
                                ByVal e As System.EventArgs) _
    Handles txtAccountNumber.Leave

    Dim Filename As String
    Dim LoanPath As String
    Dim FileFullname As String
    Dim ListOfLoans() As FileInfo
    Dim LoanReader As StreamReader
    Dim LoanFolder As DirectoryInfo
    Dim LoanInformation As FileInfo
    Dim Found As Boolean

    Found = False
    LoanPath = "C:\Watts A Loan"

    LoanFolder = New DirectoryInfo(LoanPath)
    ListOfLoans = LoanFolder.GetFiles("*", _
        SearchOption.AllDirectories)
    Filename = txtAccountNumber.Text & ".wal"
    FileFullname = LoanPath & "none.wal"

    For Each file As FileInfo In ListOfLoans
        If file.Name = Filename Then
            Found = True
            FileFullname = file.FullName
        End If
    Next

    If Found = True Then

        LoanInformation = My.Computer.FileSystem.GetFileInfo(FileFullname)
        MsgBox(LoanInformation.FullName)
        LoanReader = LoanInformation.OpenText

        Try
            txtAccountNumber.Text = LoanReader.ReadLine
            txtCustomerName.Text = LoanReader.ReadLine
            txtEmployeeNumber.Text = LoanReader.ReadLine
            txtEmployeeName.Text = LoanReader.ReadLine
            txtLoanAmount.Text = LoanReader.ReadLine
            txtInterestRate.Text = LoanReader.ReadLine
            txtPeriods.Text = LoanReader.ReadLine
            txtMonthlyPayment.Text = LoanReader.ReadLine
        Finally
            LoanReader.Close()
        End Try
    End If
End Sub

```

5. In the Class Name combo box, select btn Save

6. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub btnSave_Click(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
    Handles btnSave.Click

    Dim LoanPath As String
    Dim LoanWriter As StreamWriter

    LoanPath = "C:\Watts A Loan\" & txtAccountNumber.Text & ".wal"
    LoanWriter = My.Computer.FileSystem.OpenTextFileWriter(LoanPath, False)

    Try
        LoanWriter.WriteLine(txtAccountNumber.Text)
        LoanWriter.WriteLine(txtCustomerName.Text)
    End Try

```

```

LoanWriter.WriteLine(txtEmployeeNumber.Text)
LoanWriter.WriteLine(txtEmployeeName.Text)
LoanWriter.WriteLine(txtLoanAmount.Text)
LoanWriter.WriteLine(txtInterestRate.Text)
LoanWriter.WriteLine(txtPeriods.Text)
LoanWriter.WriteLine(txtMonthlyPayment.Text)

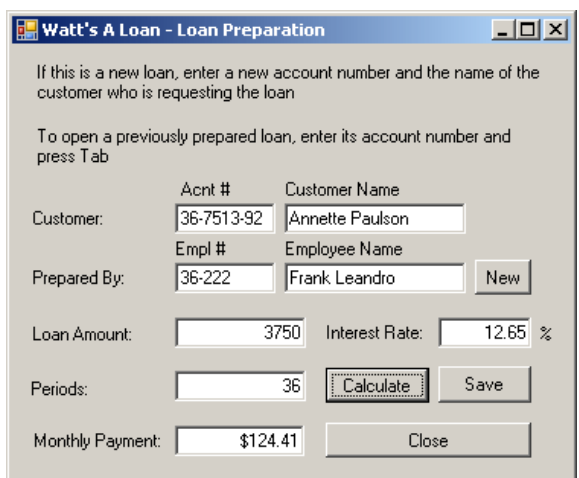
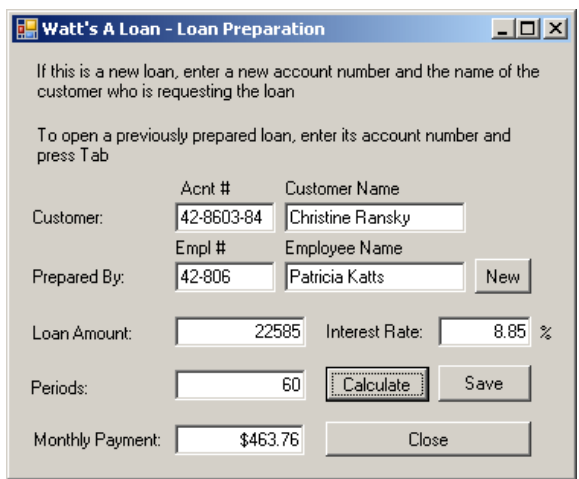
txtAccountNumber.Text = ""
txtCustomerName.Text = ""
txtEmployeeNumber.Text = ""
txtEmployeeName.Text = ""
txtLoanAmount.Text = ""
txtInterestRate.Text = ""
txtPeriods.Text = ""
txtMonthlyPayment.Text = ""
txtAccountNumber.Focus()
Finally
LoanWriter.Close()
End Try
End Sub

```

- 7. Execute the application to test it
- 8. First create a few employees as follows:

Employee #	Employee Name
42-806	Patricia Katts
75-148	Helene Mukoko
36-222	Frank Leandro
42-808	Gertrude Monay

- 9. Process a few loans



- 10. Close the application



.net Charting
FREE DEVELOPER VERSION

www.dotnetcharting.com

Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

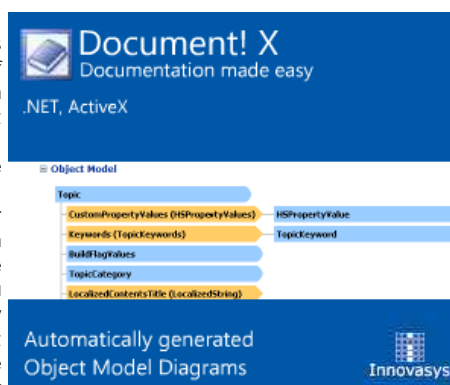


Introduction to File Processing

Overview of File Processing and Definitions

Introduction

A piece of information used in an [application](#) is primarily represented as a group of bits. So far, if we requested information from the user, when the application exited, we lost all information that the user had entered. This is because such information was only temporarily stored in the random access memory (RAM). In some cases, you will want to "keep" information that the user has entered so you can make the information available the next time the user opens the application. In some other cases, whether you request information from the user or inherently provide it to the user, you may want different people working from different [computers](#) to use or share the same data. In these and other scenarios, you must store the information somewhere and retrieve it when necessary. This is the basis of file processing.



Files

A file is a series of bytes of data that are arranged in a particular manner to produce a usable document. For easy storage, location, and management, the bytes are stored on a medium such as a [hard disc](#), a floppy disc, a compact disc, or any valid and supported type of storage. When these bytes belong to a single but common entity and hold values that are stored on a medium, the group is referred to as a file.

For greater management, files can be stored in a parent object called a directory or a folder. Since a file is a unit of storage and it stores information, it has a size, which is the number of bits it uses to store its values. To manage it, a file has a location also called a path that specifies where and/or how the file can be retrieved. Also, for better management, a file has attributes (characteristics) that indicate what can be done on the file or that provide specific information that the programmer or the [operating system](#) can use when dealing with the file.

Streams

File processing consists of creating, storing, and/or retrieving the contents of a file from a recognizable medium. For example, it is used to save word-processed files to a [hard drive](#), to store a presentation on floppy disk, or to open a file from a CD-ROM. A stream is the technique or means of performing file processing. In order to manage files stored in a computer, each file must be able to provide basic pieces of information about itself. This basic information is specified when the file is created but can change during the lifetime of a file.

To create a file, a user must first decide where it would be located: this is a requirement. A file can be located on the root drive. Alternatively, a file can be positioned inside of an existing folder. Based on security settings, a user may not be able to create a file just anywhere in the [file system](#) of the computer. Once the user has decided where the file would reside, there are various means of creating files that the users are trained to use. When creating a file, the user must give it a name following the rules of the operating system combined with those of the file system. The most fundamental piece of information a file must have is a name.

Once the user has created a file, whether the file is empty or not, the operating system assigns basic pieces of information to it. Once a file is created, it can be opened, updated, modified, renamed, etc.

Streaming Prerequisites

Introduction

To support file processing, the [.NET Framework](#) provides the **System.IO** namespace that contains many different classes to handle almost any type of file operation you may need to perform. Therefore, to perform file processing, you can include the **System.IO** namespace in


your project.

The parent class of file processing is **Stream**. With **Stream**, you can store data to a stream or you can retrieve data from a stream. **Stream** is an abstract class, which means you cannot use it to declare a variable in your application. As an abstract class, **Stream** is used as the parent of the classes that actually implement the necessary operations. You will usually use a combination of classes to perform a typical operation. For example, some classes are used to create a stream object while some others are used to write data to the created stream.

❖ Practical Learning: Introducing Streaming


1. Start [Microsoft Visual Basic](#) or Visual Studio and create a Windows Application named **ClarksvilleIceCream2**
2. In the Solution Explorer, right-click Form1.vb and click Rename
3. Type **Exercise.vb** and press Enter twice
4. In the Properties window, change the form's **Text** to **Ice Cream Vending Machine**
5. Design the form as follows:

Control	Name	Text	Additional Properties
GroupBox			
Label		Order Date:	
DateTimePicker	dtpOrderDate		Format: Short
Label		Order Time:	
DateTimePicker	dtpOrderTime		Format: Time ShowUpDown: True
Label		Flavor:	
ComboBox	cboFlavors		DropDownStyle: DropDownList
Label		Container:	
ComboBox	cboContainers		DropDownStyle: DropDownList
Label		Ingredient:	
ComboBox	cboIngredients		DropDownStyle: DropDownList
Label		Scoops:	
TextBox	txtScoops	1	TextAlign: Right
Label		Order Total:	
TextBox	txtOrderTotal	0.00	TextAlign: Right
Button	btnClose	Close	Click to end


6. Click the combo box to the right of the Flavor label. Then, in the Properties, click the ellipsis button  of **Items** property and create the list with:

Vanilla
 Cream of Cocoa
 Chocolate Chip
 Cherry Coke
 Butter Pecan
 Chocolate Cookie

Chunky Butter
Organic Strawberry
Chocolate Brownies
Caramel Au Lait

- Click OK
- Click the combo box to the right of the Container label. Then, in the Properties, click the ellipsis button  of **Items** property and create the list with:

Cone
Cup
Bowl

- Click OK
- Click the combo box to the right of the Ingredient label. Then, in the Properties, click the ellipsis button  of **Items** property and create the list with:

None
Peanuts
Mixed Nuts
M & M
Cookies

- Click OK
- Right-click the form and click View Code
- In the Class Name combo box, select txtScoops
- In the Method Name combo box, select Leave and implement the event as follows:

```
Private Sub txtScoops_Leave(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
    Handles txtScoops.Leave

    Dim PriceContainer As Double
    Dim PriceIngredient As Double
    Dim PriceScoops As Double
    Dim OrderTotal As Double
    Dim NumberOfScoops As Integer = 1

    ' The price of a container depends on which one the customer selected
    If cboContainers.Text = "Cone" Then
        PriceContainer = 0.55
    ElseIf cboContainers.Text = "Cup" Then
        PriceContainer = 0.75
    Else
        PriceContainer = 1.15
    End If

    ' Find out if the customer wants any ingredient at all
    If cboIngredients.Text = "None" Then
        PriceIngredient = 0.0
    Else
        PriceIngredient = 0.95
    End If

    Try
        ' Get the number of scoops
        NumberOfScoops = CInt(txtScoops.Text)

        If NumberOfScoops = 1 Then
            PriceScoops = 1.85
        ElseIf (NumberOfScoops = 2) Then
            PriceScoops = 2.55
        Else ' if( NumberOfScoops= 3 )
            PriceScoops = 3.25
        End If

        ' Make sure the user selected a flavor,
        ' otherwise, there is no reason to process an order
        If cboFlavors.Text <> "" Then
            OrderTotal = PriceScoops + PriceContainer + PriceIngredient
            txtOrderTotal.Text = OrderTotal.ToString("F")
        End If
    Catch ex As Exception
        MsgBox("The value you entered for the scoops is not valid" & _
            vbCrLf & "Only natural numbers such as 1," & _
            vbCrLf & " 2, or 3 are allowed" & _
            vbCrLf & "Please try again")
    End Try

End Sub
```

- Execute the application. Here is an example:

ManageEngine
ADManager Plus

**Web based
Active Directory
Reporting
&
Management
Software**

- ✔ Add / Edit Bulk User
- ✔ Mail Box Creation
- ✔ Move/Delete Bulk User
- ✔ Reset Password
- ✔ 100 Plus Reports
- ✔ Help Desk Delegation

**Starts at \$495
Download NOW!**


www.admanagerplus.com

www.admanagerplus.com
Ads by Google

16. Close the form and return to [Visual Studio](#)

The Name of a File

Before performing file processing, one of your early decisions will consist of specifying the type of operation you want the user to perform. For example, the user may want to create a brand new file, open an existing file, or perform a routine operation on a file. In all or most cases, whether you are creating a new file or manipulating an existing one, you must specify the name of the file. You can do this by declaring a **String** variable but, as we will learn later on, most classes used to create a stream can take a string that represents the file.

If you are creating a new file, there are certainly some rules you must observe. The name of a file follows the directives of the operating system. On MS DOS and Windows 3.X (that is, prior to Microsoft Windows 9X), the file had to use the 8.3 format. The actual name had to have a maximum of 8 characters with restrictions on the characters that could be used. The user also had to specify three characters after a period. The three characters, known as the file extension, were used by the operating system to classify the file. That was all necessary for those 8-bit and 16-bit operating systems. Various rules have changed. For example, the names of folders and files on Microsoft Windows >= 95 can have up to 255 characters. The extension of the file is mostly left to the judgment of the programmer but the files are still using extensions. Applications can also be configured to save different types of files; that is, files with different extensions.



At the time of this writing, the rules for file names for Microsoft Windows were on the MSDN web site at Windows Development\Windows Base Services\Files and I/O\SDK Documentation\Storage\Storage Overview\File Management\Creating, Deleting, and Maintaining Files\Naming a File (because it is a web site and not a book, its pages can change anytime).

Based on this, if you declare a **String** variable to hold the name of the file, you can simply initialize the variable with the necessary name and its extension. Here is an example:

```
Private Sub btnSave_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnSave.Click
    Dim Filename As String = "Employees.spr"
End Sub
```

❖ Practical Learning: Specifying the Name of a File

1. Right-click the form and click View Code
2. Just above the Public Class line, import the **System.IO** namespace:


```
Imports System.IO
```

```
Public Class Exercise
```

- In the Class Name combo box, select btnClose
- In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnClose_Click(ByVal sender As Object, _
                          ByVal e As System.EventArgs) _
    Handles btnClose.Click
    Dim answer As MsgBoxResult = _
        MsgBox("Do you want to save this order to remember it " & _
              "the next time you come to " & _
              "get your ice scream?", _
              MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
              "Ice Cream Vending Machine")

    If answer = MsgBoxResult.Yes Then
        Dim Filename As String = InputBox( _
            "Please type your initials and press Enter", _
            "Ice Cream Vending Machine", "AA", 100, 100)
        If Filename <> "" Then
            ' Wonderful
        Else
            MsgBox("The ice cream order will not be saved")
        End If
    End If

    MsgBox("Good Bye: It was a delight serving you")
    Close()
End Sub
```

- Scroll up in the file and, under the other using lines, type **Imports System.IO**

The Path to a File

If you declare a string as above, the file will be created in the folder as the application. Otherwise, you can create your new file anywhere in the hard drive. To do that, you must provide a complete path where the file will reside. A path is a string that specifies the drive (such as A:, C:, or D:). The sections of a complete path string are separated by a backslash. For example, a path can be made of a folder followed by the name of the file. An example would be

```
C:\Palermo.tde
```

A path can also consist of a drive followed by the name of the folder in which the file will be created. Here is an example:

```
C:\Program Files\Palermo.tde
```

A path can also indicate that the file will be created in a folder that itself is inside of another folder. In this case, remember that the names of folder must be separated by backslashes.

When providing a path to the file, you could encounter different types of problems:

- If the drive you specify does not exist or cannot be read, the compiler would consider that the file does not exist
- If you provide folders that do not exist in the drive, the compiler would consider that the file does not exist. This also means that the compiler will not create the folder(s) (the .NET Framework provides all means to create a folder but you must ask the compiler to create it; simply specifying a folder that does not exist will not automatically create it, even if you are creating a new file).

Therefore, it is your responsibility to make sure that either the file or the path to the file is valid. As we will see in the next sections, the compiler can check the existence of a file or path.

eWebGuru.com Reliable Web Hosting	2 GB Web Space 20 GB Bandwidth Unlimited Email Account	24X7 Live Support 99.9% Uptime cPanel	Rs. 995/Yr

eWebGuru.com/Web_Hosting_India

Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



File Processing

The .NET Support for Files

Introduction

The primary support of a file as an object is provided by a .NET Framework class called **File**. This **Shared** class is equipped with various types of (static) methods to create, save, open, copy, move, delete, or check the existence of a file. As an alternative, the Microsoft Visual Basic library provides the **My** object that includes the **FileSystem** object in the **Computer** object.

www.manashosting.com

Arts by Google

File Existence

One of the valuable operations that the **File** class can perform is to check the existence of the file you want to use. For example, if you are creating a new file, you may want to make sure it does not exist already because if you try to create a file that exists already, the compiler may first delete the old file before creating the new one. This could lead to unpredictable results, especially because such a file is not sent to the Recycle Bin. On the other hand, if you are trying to open a file, you should first make sure the file exists, otherwise the compiler will not be able to open a file it cannot find.

To check the existence of a file, the **File** class provides the **Exists** method. Its syntax is:

```
Public Shared Function Exists(path As String) As Boolean
```

To perform this operation using the **FileSystem** class from **My**, you can call its **FileExists()** method whose syntax is:

```
Public Function FileExists(ByVal file As String) As Boolean
```

If you provide only the name of the file, the compiler would check it in the folder of the application. If you provide the path to the file, the compiler would check its drive, its folder(s) and the file itself. In both cases, if the file exists, the method returns **True**. If the compiler cannot find the file, the method returns **False**. It is important to know that if you provide a complete path to the file, any slight mistake would produce a **False** result.

File Creation

Besides checking the existence of the file, the **File** class can be used to create a new file. To support this operation, the **File** class is equipped with the **Create()** method that is overloaded with two versions as follows:

```
Public Shared Function Create(path As String) As FileStream
Public Shared Function Create(path As String, bufferSize As Integer) As FileStream
```

In both cases, the **File.Create()** method returns a **Stream** value, which is a **FileStream** value. As the **File.Create()** method indicates, it takes the name or path of the file as argument. If you know or want to specify the size, in bytes, of the file, you can use the second version.

To provide the same operation of creating a file, you can use the **Open()** method of the **File** class. It is overloaded in three versions as follows:

```
Public Shared Function Open ( _
    path As String, _
    mode As FileMode _
```

```

) As FileStream

Public Shared Function Open ( _
    path As String, _
    mode As FileMode, _
    access As FileAccess _
) As FileStream
Public Shared Function Open ( _
    path As String, _
    mode As FileMode, _
    access As FileAccess, _
    share As FileShare _
) As FileStream

```

Access to a File

In order to perform an operation on a file, you must specify to the operating system how to proceed. One of the options you have is to indicate the type of access that will be granted on the file. This access is specified using the **FileAccess** enumerator. The members of the **FileAccess** enumerator are:

- **FileAccess.Write**: New data can be written to the file
- **FileAccess.Read**: Existing data can be read from the file
- **FileAccess.ReadWrite**: Existing data can be read from the file and new data be written to the file

Ads by Google



[Convert Text to XML](#)

Convert any Text Format to XML Java and .NET Converter Components

www.xmlconverters.com/Convert-T

File Sharing

In standalone workstations, one person is usually able to access and open a file then perform the necessary operations on it. In networked computers, you may create a file that different people can access at the same time or you may make one file access another file to retrieve information. For example, suppose you create an application for a fast food restaurant that has two or more connected workstations and all workstations save their customers orders to a common file. In this case, you must make sure that any of the computers can access the file to save an order. An employee from one of these workstations must also be able to open the file to retrieve a customer order for any necessary reason. You can also create a situation where one file holds an inventory of the items of a store and another file holds the customers orders. Obviously one file would depend on another. Based on this, when an operation must be performed on a file, you may have to specify how a file can be shared. This is done through the **FileShare** enumerator.

The values of the **FileShare** enumerator are:

- **FileShare.Inheritable**: Allows other file handles to inherit from this file
- **FileShare.None**: The file cannot be shared
- **FileShare.Read**: The file can be opened and read from
- **FileShare.Write**: The file can be opened and written to
- **FileShare.ReadWrite**: The file can be opened to write to it or read from it

[C/C++ Programmers needed](#)

Join GetAFreelancer.com and bid on projects. Free and quick signup.

www.GetAFreelancer.com

[PC based Instruments](#)

Data Acquisition, Signal Generation and Digital I/O with LabView driver

www.spectrum-instrumentation.com

[Object COBOL Transition](#)

Upgrade your COBOL applications using Micro Focus Server Express

www.microfocus.com/hpconversion

The Mode of a File

Besides the access to the file, another option you will most likely specify to the operating system is referred to as the mode of a file. It is specified through the **FileMode** enumerator. The members of the **FileMode** Enumerator are:

- **FileMode.Append**: If the file already exists, the new data will be added to its end. If the file doesn't exist, it will be created and the new data will be added to it
- **FileMode.Create**: If the file already exists, it will be deleted and a new file with the same name will be created. If the file doesn't exist, then it will be created
- **FileMode.CreateNew**: If the new already exists, the compiler will throw an error. If the file doesn't exist, it will be created
- **FileMode.Open**: If the file exists, it will be opened. If the file doesn't exist, an error would be thrown
- **FileMode.OpenOrCreate**: If the file already exists, it will be opened. If the file doesn't exist, it will be created
- **FileMode.Truncate**: If the file already exists, its contents will be deleted completely but the file will be kept, allowing you to write new data to it. If the file doesn't exist, an error would be thrown

[Snap Stream](#)

Compare prices at mySimon - winner of "Best Internet Shopping Service"

mySimon.com

Starting and Closing a Stream

Starting a Stream

File streaming consists of performing one of the routine operations on a file, such as creating or opening it. This basic operation can be performed using a class called **FileStream**. You can use a **FileStream** object to get a stream ready for processing. As one of the most complete classes of file processing of the .NET Framework, **FileStream** is equipped with all necessary properties and methods. To use it, you must first declare a variable of it. The class is equipped with nine constructors.

One of the constructors of the **FileStream** class has the following syntax:

```
Public Sub New(path As String, mode As FileMode)
```

This constructor takes as its first argument the name of the file or its path. The second argument specifies the type of operation to perform on the file. Here is an example of calling this method:

```
Imports System.IO
```

```
Public Class Exercise
```

```
    Private Sub btnSave_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
        Handles btnSave.Click
        Dim Filename As String = "Persons.prs"
```

```
        Dim StreamPersons As FileStream = New FileStream(Filename, FileMode.Create)
```

```
    End Sub
```

```
End Class
```

Closing a Stream

When you use a stream, it requests resources from the operating system and uses them while the stream is available. When you are not using the stream anymore, you should free the resources and make them available again to the operating system so that other services can use them. This is done by closing the stream.

To close a stream, you can call the **Close()** method of the class(es) you were using. Here are examples:

```
Public Class Exercise
```

```
    Private Sub btnSave_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
        Handles btnSave.Click
        Dim Filename As String = "Persons.prs"
```

```
        Dim StreamPersons As FileStream = New FileStream(Filename, FileMode.Create)
```

```
        StreamPersons.Close()
```

```
    End Sub
```

```
End Class
```

The image is a horizontal banner. On the left is a blue box for 'Document! X' with the tagline 'Documentation made easy' and lists '.NET, ActiveX, Database Schemas, XSD Schemas, VB6, VBA'. In the center is a screenshot of a code editor showing VB.NET code for a 'Customers' class. On the right is a blue box with a testimonial: 'Chosen by industry leaders including Infragistics, Dundas, Telerik, Farpoint, Data Dynamics, Janus Systems' and the Innovasys logo. Below the testimonial is the text 'Feedback - Ads by Google'.

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



File Processing: Reading From a Stream

Binary Reading

As opposed to writing to a stream, you may want to read existing data from it. Before doing this, you can first specify your intent to the streaming class using the **FileMode** enumerator. This can be done using the **FileStream** class as follows:

```
Private Sub btnOpen_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnOpen.Click

    Dim filename As String = "Persons.prs"

    Dim PersonsStreamer As FileStream = _
        New FileStream(filename, FileMode.Create)

End Sub
```

[ECO for Visual Studio](#)

Much more than just an ORM Download free version today
[capableobjects.com](#)

[Reading Pen for Dyslexia](#)

\$244 School Purchase Order H.Q. Only \$244 after \$35 Instant Rebate
[www.quicktionarysuperstore.com](#)

[Nitrogen Management](#)

GreenSeeker variable rate and mapping systems.
[www.aritytech.com](#)



Ads by Google

Once the stream is ready, you can get prepared to read data from it. To support this, you can use the **BinaryReader** class. This class provides two constructors. One of the constructors (the first) has the following syntax:

```
Public Sub New(input As Stream)
```

This constructor takes as argument a **Stream** value, which could be a **FileStream** object. After declaring a **FileStream** variable using this constructor, you can read data from it. To support this, the class provides an appropriate method for each primitive data type.

After using the stream, you should close it to reclaim the resources it was using. This is done by calling the **Close()** method.

Here is an example of using the mentioned methods:

```
Private Sub btnOpen_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnOpen.Click

    Dim filename As String = "Persons.prs"
    Dim PersonsStreamer As FileStream
    Dim PersonsReader As BinaryReader

    PersonsStreamer = New FileStream(filename, FileMode.Open)
    PersonsReader = New BinaryReader(PersonsStreamer)

    txtPerson1.Text = PersonsReader.ReadString()
    txtPerson2.Text = PersonsReader.ReadString()
    txtPerson3.Text = PersonsReader.ReadString()
    txtPerson4.Text = PersonsReader.ReadString()

    PersonsReader.Close()
    PersonsStreamer.Close()

End Sub
```

Stream Reading

Besides the **BinaryReader** class that reads its values in binary format, the .NET Framework supports character reading through a class called **StreamReader**. The **StreamReader** class is based on the **TextReader** class. Like its counterpart the **StreamWriter** class, **StreamReader** is equipped with various constructors. If you want to read values from a file, you can pass its name or path to the following constructor:

```
Public Sub New(path As String)
```

Alternatively, to a **FileStream** object, you can pass it to the following constructor of the **StreamReader** class to create a stream:

```
Public Sub New (stream As Stream)
```

This method takes as argument a **Stream**-based variable, which could be a **FileStream** value. If/since you are planning to read from a stream, configure your file mode status appropriately. Here is an example:

```
Private Sub btnOpen_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
    Handles btnOpen.Click
    Dim Filename As String = "Student.std"

    Dim StudentsStreamer As FileStream
    Dim StudentsReader As StreamReader

    StudentsStreamer = New FileStream(Filename, FileMode.Open)
    StudentsReader = New StreamReader(StudentsStreamer)
End Sub
```

After creating a **StreamReader** object, you can read data from the file. To support this, the **TextReader** class is equipped with the **Write()** method that the **StreamReader** class inherits. The **StreamReader** class itself is equipped with the **ReadLine()** method. Here is an example of calling it:

```
Private Sub btnOpen_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
    Handles btnOpen.Click
    Dim Filename As String = "Student.std"

    Dim StudentsStreamer As FileStream
    Dim StudentsReader As StreamReader

    StudentsStreamer = New FileStream(Filename, FileMode.Open)
    StudentsReader = New StreamReader(StudentsStreamer)

    txtFirstName.Text = StudentsReader.ReadLine()
    txtLastName.Text = StudentsReader.ReadLine()
    cbxGenders.SelectedIndex = CInt(StudentsReader.ReadLine())

    StudentsReader.Close()
    StudentsStreamer.Close()
End Sub
```

Using My File System

To read text from a file using the **FileSystem** class from **My** object, you can call the **OpenTextFileReader()** method. It comes in two versions whose syntaxes are:

```
Public Shared Function OpenTextFileReader( _
    ByVal file As String,
) As System.IO.StreamReader

Public Shared Function OpenTextFileReader( _
    ByVal file As String, _
    ByVal encoding As System.Text.Encoding _
) As System.IO.StreamReader
```

The first argument is the name of, or the path to, the file that will receive the new values. The *encoding* argument allows you to specify the type of text that you want to use.

This method returns a **StreamReader** value. After creating the stream reader, you can then read values from it. To do this, you can call the **ReadLine()** method of the **StreamReader** class. Once again, after using the stream reader, remember to close it. Here are examples:

```
Private Sub Button2_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
    Handles Button2.Click
    Dim fstPeople As StreamReader

    fstPeople = My.Computer.FileSystem.OpenTextFileReader("People1.ppl")

    txtPerson1.Text = fstPeople.ReadLine
    txtPerson2.Text = fstPeople.ReadLine
    txtPerson3.Text = fstPeople.ReadLine
    txtPerson4.Text = fstPeople.ReadLine

    fstPeople.Close()
End Sub
```

❖ Practical Learning: Reading From a Stream

1. Access the Code and, in the Class Name combo box, select (Exercise Events)
2. In the Method Name combo box, select Load and implement the event as follows:



```

Private Sub Exercise_Load(ByVal sender As Object, _
                        ByVal e As System.EventArgs) _
                        Handles Me.Load

    Dim OrderDate As String, OrderTime As String
    Dim SelectedFlavor As String
    Dim SelectedContainer As String
    Dim SelectedIngredient As String
    Dim Scoops As String, OrderTotal As String
    Dim Filename As String
    Dim IceCreamReader As StreamReader

    Filename = InputBox( _
        "If you had previously ordered an ice cream here " & _
        "and you want to order the same, please type your " & _
        "initials and press Enter (otherwise, press Esc)", _
        "Ice Cream Vending Machine", "", 100, 100)

    If Filename <> "" Then
        Filename = Filename & ".icr"

        IceCreamReader = My.Computer.FileSystem.OpenTextFileReader(Filename)

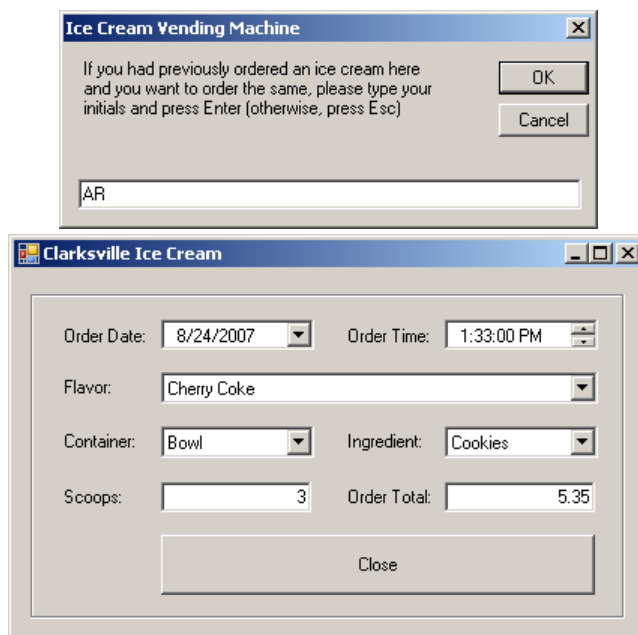
        ' Find out if this order was previously saved in the machine
        If My.Computer.FileSystem.FileExists(Filename) Then
            ' If so, open it
            OrderDate = IceCreamReader.ReadLine()
            OrderTime = IceCreamReader.ReadLine()
            SelectedFlavor = IceCreamReader.ReadLine()
            SelectedContainer = IceCreamReader.ReadLine()
            SelectedIngredient = IceCreamReader.ReadLine()
            Scoops = IceCreamReader.ReadLine()
            OrderTotal = IceCreamReader.ReadLine()

            ' And display it to the user
            dtpOrderDate.Value = DateTime.Parse(OrderDate)
            dtpOrderTime.Value = DateTime.Parse(OrderTime)
            cboFlavors.Text = SelectedFlavor
            cboContainers.Text = SelectedContainer
            cboIngredients.Text = SelectedIngredient
            txtScoops.Text = Scoops.ToString()
            txtOrderTotal.Text = OrderTotal

            IceCreamReader.Close()
        Else
            MsgBox("It looks like you have not previously " & _
                "ordered an ice cream here")
        End If
    End If
End Sub

```

3. Execute the application and test it. Here is an example:



4. Close the form

Unlimited Webspace and Unlimited Bandwidth **FREE** **Only**
Domain Registration **Rs. 2000/-**

- ⇒ 100 Email Ids of 10 GB
- ⇒ All Database FREE
- ⇒ 5000 Templates / Website Builder
- ⇒ Full Control Panel
- ⇒ 200 SEO Tools/Softwares/Weblinks

www.manashosting.com Feedback - Ads by Google

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Home](#)



File Processing: Writing to a Stream

Binary Writing

A streaming operation is typically used to create a stream. Once the stream is ready, you can write data to it. The writing operation is performed through various classes. One of these classes is called **BinaryWriter**.

The **BinaryWriter** class can be used to write values of primitive data types (**char**, **int**, **float**, **double**, etc). To use a **BinaryWriter** value, you can first declare its variable. To do this, you would use one of the class' three constructors. The first constructor is the default. The second constructor has the following syntax:



```
Public Sub New(output As Stream)
```

This constructor takes as argument a **Stream** value, which could be a **FileStream** variable. Here is an example:

```
Public Class Exercise
```

```
Private Sub btnSave_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles btnSave.Click
    Dim Filename As String = "Persons.prs"

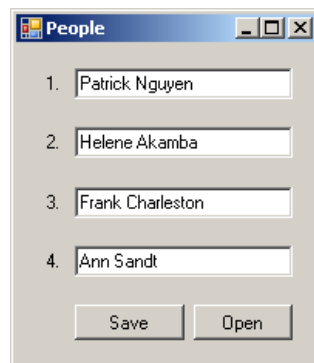
    Dim StreamPersons As FileStream = New FileStream(Filename, FileMode.Create)
    Dim WriterPersons As BinaryWriter = new BinaryWriter(fstPersons)
```

```
End Sub
```

```
End Class
```

As mentioned already, make sure you close a stream after using it. In the same way, make sure you close a writer after using it. If you are using both, you should close them in the reverse order they were used.

Most classes that are used to add values to a stream are equipped with a method called **Write**. This is also the case for the **BinaryWriter** class. This method takes as argument the value that must be written to the stream. The method is overloaded so that there is a version for each primitive data type. Here is an example that adds strings to a newly created file:



```
Imports System.IO
```

```
Public Class Exercise
```

```
Private Sub btnSave_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
    Handles btnSave.Click
```

```

Dim Filename As String = "Persons.prs"

Dim PersonsStream As FileStream
Dim PersonsWriter As BinaryWriter

PersonsStream = New FileStream(Filename, FileMode.Create)
PersonsWriter = New BinaryWriter(PersonsStream)

PersonsWriter.Write(txtPerson1.Text)
PersonsWriter.Write(txtPerson2.Text)
PersonsWriter.Write(txtPerson3.Text)
PersonsWriter.Write(txtPerson4.Text)

PersonsWriter.Close()
PersonsStream.Close()

txtPerson1.Text = ""
txtPerson2.Text = ""
txtPerson3.Text = ""
txtPerson4.Text = ""
End Sub
End Class

```

Stream Writing

As mentioned already, the **BinaryWriter** class considers the values it has to write as binary values in the orders of integers, characters, and their variants. In some cases, you may want to write a block of text. To support this, the .NET Framework provides the **StreamWriter** class. **StreamWriter** is derived from the **TextWriter** class, which the base class used to write a series of characters to a stream. To assist you with writing operations, the **StreamWriter** class is equipped with various constructors.

If you had previously defined a **FileStream** object and you want to write values to it, you can use the following constructor of the **StreamWriter** class to create a stream:

```
Public Sub New (stream As Stream)
```

This method takes as argument a **Stream**-based variable, which could be a **FileStream** value. Here is an example of using it:

```
Imports System.IO

Public Class StudentRegistration

    Private Sub btnSave_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
        Handles btnSave.Click
        Dim Filename As String = "Student.std"

        Dim StudentsStreamer As FileStream
        Dim StudentsWriter As StreamWriter

        StudentsStreamer = New FileStream(Filename, FileMode.Create)
        StudentsWriter = New StreamWriter(StudentsStreamer)
    End Sub
End Class

```

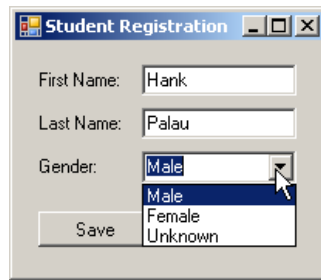
If you do not want to use a Stream-based class, you can directly provide a file to the **StreamWriter**. To support this, the class is equipped with the following constructor:

```
Public Sub New(path As String)
```

This method takes as argument the name of, or a path to, a file.

After creating a **StreamWriter** object, you can write one or more values to it. To support this, the **TextWriter** class is equipped with the **Write()** method that the **StreamWriter** class inherits. The **StreamWriter** class itself is equipped with a method named **WriteLine** that is given in various versions, each version adapted to a particular data type. The **Write()** method writes text on a line and keeps the caret on the same line. The **WriteLine()** method writes a line of text and moves the caret to the next line.

Here is an example of using a **StreamWriter** class to write values to a file:



```
Imports System.IO
```

```
Public Class StudentRegistration
```

```
    Private Sub btnSave_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
        Handles btnSave.Click
        Dim filename As String = "Student.std"

        Dim StudentsStreamer As FileStream
        Dim StudentsWriter As StreamWriter

        StudentsStreamer = New FileStream(filename, FileMode.Create)
        StudentsWriter = New StreamWriter(StudentsStreamer)

        StudentsWriter.WriteLine(txtFirstName.Text)
        StudentsWriter.WriteLine(txtLastName.Text)
        StudentsWriter.WriteLine(cbxGenders.SelectedIndex)

        StudentsWriter.Close()
        StudentsStreamer.Close()

        txtFirstName.Text = ""
        txtLastName.Text = ""
        cbxGenders.SelectedIndex = 2
    End Sub
```

```
End Class
```

Using My File System

By default, the **FileStream** class does not specify what operation is going to be performed on the file. If you are planning to create a new file and write values to it, you can call the **OpenTextWriter()** method of the **FileSystem** class of the **My** object. It comes in two versions whose syntaxes are:

```
Public Function OpenTextWriter( _
    ByVal file As String, _
    ByVal append As Boolean _
) As System.IO.StreamWriter

Public Function OpenTextWriter( _
    ByVal file As String, _
    ByVal append As Boolean, _
    ByVal encoding As System.Text.Encoding _
) As System.IO.StreamWriter
```

The first argument is the name of, or the path to, the file that will receive the new values. The second argument specifies whether this is a new file or you are opening it to add new values to it. If this is a new file, pass the *append* argument as **False**. The *encoding* argument allows you to specify the type of text that you want to use.

This method returns a **StreamWriter**. Here is an example of creating a stream writer by calling the **OpenTextWriter()** method:

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
        Handles Button1.Click
        Dim fstPeople As StreamWriter

        fstPeople = My.Computer.FileSystem.OpenTextWriter("People1.ppl", False)

    End Sub
End Class
```

After creating the stream writer, you can then write values to it. To do this, you can call the **WriteLine()** method of the **StreamWriter** class. Once again, after using the stream writer,

MANASHOSTING
MANASHOSTING

Unlimited Pack

- One Domain FREE
- Unlimited Webspace
- Unlimited Bandwidth
- 100 E-mail Ids
- 5000 Templates
- Website Builder
- 500 Source Code
- 200 SEO Tools
- 100 Softwares

Only Rs.2000/-
Per Year

VIEW PLANS >

www.manashosting.com
Ads by Google

remember to close it. Here are examples:

```
Imports System.IO

Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) _
        Handles Button1.Click
        Dim fstPeople As StreamWriter

        fstPeople = My.Computer.FileSystem.OpenTextFileWriter("People1.ppl", False)

        fstPeople.WriteLine(txtPerson1.Text)
        fstPeople.WriteLine(txtPerson2.Text)
        fstPeople.WriteLine(txtPerson3.Text)
        fstPeople.WriteLine(txtPerson4.Text)
        fstPeople.Close()

        txtPerson1.Text = ""
        txtPerson2.Text = ""
        txtPerson3.Text = ""
        txtPerson4.Text = ""
    End Sub
End Class
```

❖ Practical Learning: Writing to a Stream

1. Scroll back down the file and change the code of the btnClose_Click event as follows:

```
Private Sub btnClose_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnClose.Click

    Dim Filename As String
    Dim Answer As MsgBoxResult
    Dim IceCreamWriter As StreamWriter

    Answer = MsgBox("Do you want to save this order to remember it " & _
        "the next time you come to " & _
        "get your ice scream?", _
        MsgBoxStyle.YesNo Or MsgBoxStyle.Question, _
        "Ice Cream Vending Machine")
    If Answer = MsgBoxResult.Yes Then
        Filename = InputBox( _
            "Please type your initials and press Enter", _
            "Ice Cream Vending Machine", "AA", 100, 100)
        If Filename <> "" Then
            Filename = Filename & ".icr"

            IceCreamWriter = _
                My.Computer.FileSystem.OpenTextFileWriter(Filename, False)

            IceCreamWriter.WriteLine(dtpOrderDate.Value.ToShortDateString())
            IceCreamWriter.WriteLine(dtpOrderTime.Value.ToShortTimeString())
            IceCreamWriter.WriteLine(cboFlavors.Text)
            IceCreamWriter.WriteLine(cboContainers.Text)
            IceCreamWriter.WriteLine(cboIngredients.Text)
            IceCreamWriter.WriteLine(txtScoops.Text)
            IceCreamWriter.WriteLine(txtOrderTotal.Text)

            IceCreamWriter.Close()

            MsgBox("The order has been saved")
        Else
            MsgBox("The ice cream order will not be saved")
        End If
    End If

    MsgBox("Good Bye: It was a delight serving you")
    Close()
End Sub
```

2. Execute the application and create an ice cream order. Here is an example:

Ice Cream Vending Machine

Order Date: 4/18/2008 Order Time: 2:15:20 PM

Flavor: Chunky Butter

Container: Cone Ingredient: Peanuts

Scoops: 2 Order Total: 4.05

Close

Ice Cream Vending Machine

Order Date: 4/18/2008 Order Time: 2:15:20 PM

Flavor: Chunky Butter

Container: Cone Ingredient: Peanuts

Scoops: 2 Order Total: 4.05

Close

Ice Cream Vending Machine

Do you want to save this order to remember it the next time you come to get your ice cream?

Yes No

Ice Cream Vending Machine

Order Date: 4/18/2008 Order Time: 2:15:20 PM

Flavor: Chunky Butter

Container: Cone Ingredient: Peanuts

Scoops: 2 Order Total: 4.05

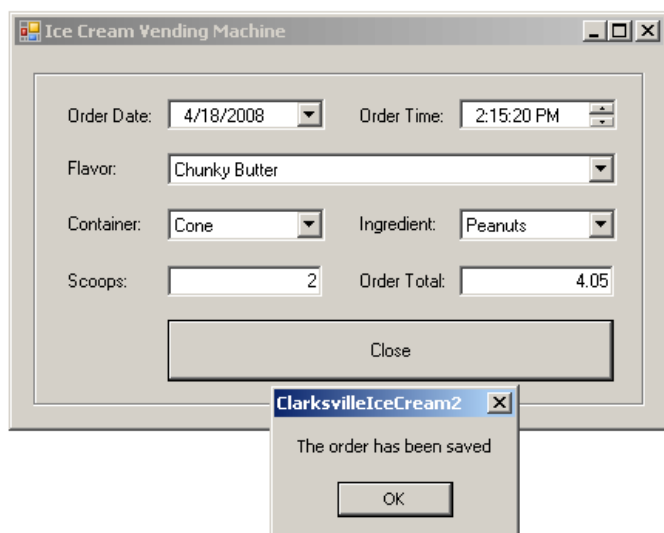
Close

Ice Cream Vending Machine

Please type your initials and press Enter

OK Cancel

JD



3. Close the form and return to your programming environment

Looking for a new Property?	Over 2,00,000 properties	CLICK HERE >	 No.1 Property Portal
For Sale	now available		
Apartment, 3 bedrooms Rs.27,85,000 Only!			

[Previous](#)

Copyright © 2008 FunctionX, Inc.

[Next](#)



Binary Serialization

Object Serialization and De-Serialization

Introduction

Consider the following program:



```
Imports System.IO
```

```
Public Class Exercise
```

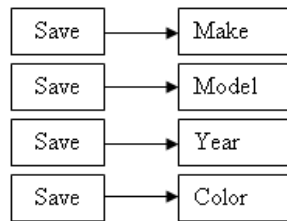
```
Private Sub btnWrite_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnWrite.Click
    Dim Make As String = txtMake.Text
    Dim Model As String = txtModel.Text
    Dim Year As Integer = CInt(txtYear.Text)
    Dim CarColor As Integer = cbxColors.SelectedIndex

    Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Create)
    Dim bnwCar As BinaryWriter = New BinaryWriter(stmCar)

    Try
        bnwCar.Write(Make)
        bnwCar.Write(Model)
        bnwCar.Write(Year)
        bnwCar.Write(CarColor)
    Finally
        bnwCar.Close()
        stmCar.Close()
    End Try
End Sub
End Class
```

Here is an example of running the program:

This is an example of the techniques used in file processing to save [individual](#) data of primitive types:



The values can be retrieved with the following code:

```

Imports System.IO

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim Make As String = txtMake.Text
        Dim Model As String = txtModel.Text
        Dim Year As Integer = CInt(txtYear.Text)
        Dim CarColor As Integer = cbxColors.SelectedIndex

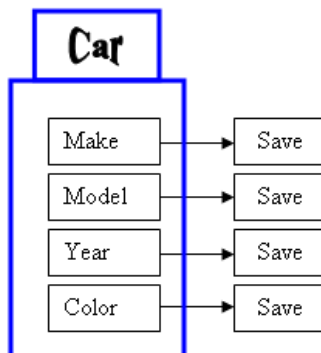
        Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Create)
        Dim bnwCar As BinaryWriter = New BinaryWriter(stmCar)

        Try
            bnwCar.Write(Make)
            bnwCar.Write(Model)
            bnwCar.Write(Year)
            bnwCar.Write(CarColor)
        Finally
            bnwCar.Close()
            stmCar.Close()
        End Try
    End Sub

    Private Sub btnRead_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnRead.Click
        Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Open)
        Dim bnrCar As BinaryReader = New BinaryReader(stmCar)

        Try
            txtMake.Text = bnrCar.ReadString()
            txtModel.Text = bnrCar.ReadString()
            txtYear.Text = bnrCar.ReadUInt32().ToString()
            cbxColors.SelectedIndex = bnrCar.ReadInt32()
        Finally
            bnrCar.Close()
            stmCar.Close()
        End Try
    End Sub
End Class
  
```

In the same way, you can save the individual fields of a class or you can retrieve the individual fields of a car:



Here is an example:

Class: **Car.vb**

```

Public Class Car
    Public Make As String
    Public Model As String
    Public Year As Integer
  
```



```

    Public Color As Integer
End Class

Imports System.IO

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim Vehicle As Car = New Car()

        Vehicle.Make = txtMake.Text
        Vehicle.Model = txtModel.Text
        Vehicle.Year = CInt(txtYear.Text)
        Vehicle.Color = cbxColors.SelectedIndex

        Dim stmCar As FileStream = New FileStream("Car2.car", FileMode.Create)
        Dim bnwCar As BinaryWriter = New BinaryWriter(stmCar)

        Try
            bnwCar.Write(Vehicle.Make)
            bnwCar.Write(Vehicle.Model)
            bnwCar.Write(Vehicle.Year)
            bnwCar.Write(Vehicle.Color)
        Finally
            bnwCar.Close()
            stmCar.Close()
        End Try
    End Sub

    Private Sub btnRead_Click(ByVal sender As Object, _
                              ByVal e As System.EventArgs) Handles btnRead.Click
        Dim stmCar As FileStream = New FileStream("Car2.car", FileMode.Open)
        Dim bnrCar As BinaryReader = New BinaryReader(stmCar)

        Try
            Dim Vehicle As Car = New Car()

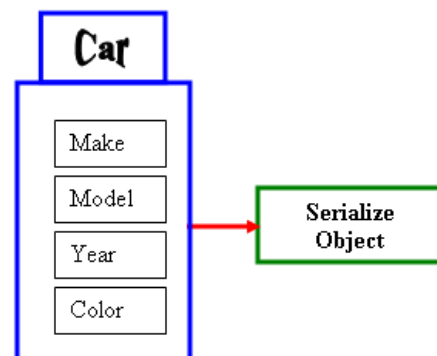
            Vehicle.Make = bnrCar.ReadString()
            Vehicle.Model = bnrCar.ReadString()
            Vehicle.Year = bnrCar.ReadUInt32()
            Vehicle.Color = bnrCar.ReadInt32()

            txtMake.Text = Vehicle.Make
            txtModel.Text = Vehicle.Model
            txtYear.Text = Vehicle.Year
            cbxColors.SelectedIndex = Vehicle.Color
        Finally
            bnrCar.Close()
            stmCar.Close()
        End Try
    End Sub
End Class

```

When it comes to a class, the problem with saving individual fields is that you could forget to save one of the fields. For example, considering a Car class, if you don't save the Make information of a Car object and retrieve or open the saved object on another [computer](#), the receiving user would miss some information and the car cannot be completely identifiable. An alternative is to save the whole Car object.

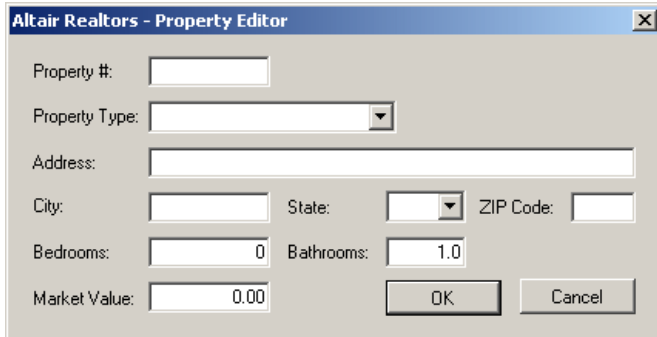
Object serialization consists of saving a whole object as one instead of its individual fields:



In other words, a variable declared from a class can be saved to a stream and then the saved object can be retrieved later or on another computer. The [.NET Framework](#) supports two types of object serialization: binary and SOAP.

❖ Practical Learning: Introducing Serialization

1. Start Microsoft Visual Basic and create a Windows Forms [Application](#) named **RealEstate1**
2. To create a new form, on the main menu, click Projects -> Add Windows Form...
3. Set the Name to **PropertyEditor** and click Add
4. Design the form as follows:

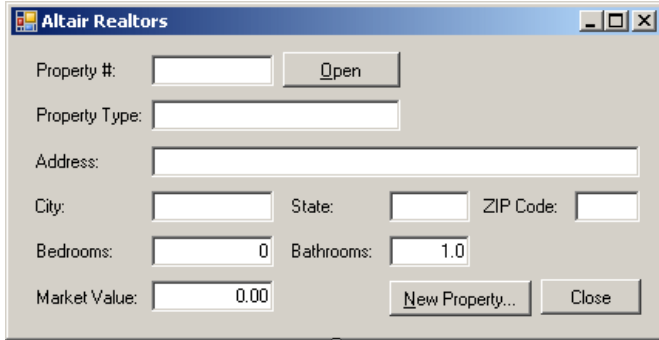


Control	Text	Name	Other Properties
Label	A Property #:		
TextBox	abl	txtPropertyNumber	Modifiers: Public
Label	A Property Type:		
ComboBox	abl	cbxPropertyTypes	Modifiers: Public Items: Unknown Single Family Townhouse Condominium
Label	A Address:		
TextBox	abl	txtAddress	Modifiers: Public
Label	A City:		
TextBox	abl	txtCity	Modifiers: Public
Label	A State:		
ComboBox	abl	cbxStates	Modifiers: Public Items: DC MD PA VA WV
Label	A ZIP Code:		
TextBox	abl	txtZIPCode	Modifiers: Public
Label	A Bedrooms:		
TextBox	abl 0	txtBedrooms	Modifiers: Public
Label	A Bathrooms:		
TextBox	abl 1.0	txtBathrooms	Modifiers: Public
Label	A Market Value:		
TextBox	abl 0.00	txtMarketValue	Modifiers: Public
Button	abl OK	btnOK	DialogResult: OK
Button	abl Cancel	btnCancel	DialogResult: Cancel

Form

FormBorderStyle: FixedDialog
 Text: Altair [Realtors](#) - Property Editor
 StartPosition: CenterScreen
 AcceptButton: btnOK
 CancelButton: btnCancel
 MaximizeBox: False
 MinimizeBox: False
 ShowInTaskBar: False

5. In the Solution Explorer, right-click Form1.vb and click Rename
6. Type **RealEstate.vb** and press Enter twice (to display that form)
7. Design the form as follows:



Control	Text	Name
Label	A Property #:	
TextBox		txtPropertyNumber
Button	Open	btnOpen
Label	A Property Type:	
TextBox		txtPropertyType
Label	A Address:	
TextBox		txtAddress
Label	A City:	
TextBox		txtCity
Label	A State:	
TextBox		txtState
Label	A ZIP Code:	
TextBox		txtZIPCode
Label	A Bedrooms:	
TextBox	0	txtBedrooms
Label	A Bathrooms:	
TextBox	1.0	txtBathrooms
Label	A Market Value:	
TextBox	0.00	txtMarketValue
Button	&New Property...	btnNewProperty
Button	Close	btnClose

Form

Ads by Google

[Remove Excel Duplicates](#)

Find and remove duplicated columns, rows and cells in Excel 2000-2007.

www.office-excel.com

```

FormBorderStyle: FixedDialog
Text: Altair Realtors - Property Editor
StartPosition: CenterScreen

```

8. Save the form

Binary Serialization

Binary serialization works by processing an object rather than streaming its individual member variables. This means that, to use it, you define an object and initialize it, or "fill" it, with the necessary values and any information you judge necessary. This creates a "state" of the object. It is this state that you prepare to serialize. When you save the object, it is converted into a stream.

To perform binary serialization, there are a few steps you must follow. When creating the class whose objects would be serialized, start it with the **<Serializable>** attribute. Here is an example:

```

<Serializable()> Public Class Car
    Public Make As String
    Public Model As String
    Public Year As Integer
    Public Color As Integer
End Class

```

Before serializing an object, you should reference the **System.Runtime.Serialization.Formatters.Binary** namespace. The class responsible for binary serialization is called **BinaryFormatter**. This class is equipped with two constructors. The default constructor is used to simply create an object.

After declaring the variable, to actually serialize an object, call the **Serialize()** method of the **BinaryFormatter** class. The method is overloaded with two versions. One of the versions of this method uses the following syntax:

```

Public Sub Serialize ( _
    serializationStream As Stream, _
    graph As Object _
)

```

The first argument to this method must be an object of a **Stream**-based class, such as a **FileStream** object. The second argument must be the object to serialize. This means that, before calling this method, you should have built the object.

Here is an example:

```

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim vehicle As Car = New Car()
        vehicle.Make = txtMake.Text
        vehicle.Model = txtModel.Text
        vehicle.Year = CInt(txtYear.Text)
        vehicle.Color = cbxColors.SelectedIndex

        Dim stmCar As FileStream = New FileStream("Car3.car", FileMode.Create)
        Dim bfmCar As BinaryFormatter = New BinaryFormatter()

        bfmCar.Serialize(stmCar, vehicle)
    End Sub
End Class

```

❖ Practical Learning: Serializing an Object

1. To create a new class, on the main menu, click Project -> Add Class...
2. Set the Name to **SampleProperty** and click Add
3. Change the file as follows:

```

<Serializable()> Public Class SampleProperty
    Public PropertyNumber As String
    Public PropertyType As String
    Public Address As String
    Public City As String
    Public State As String

```

```

Public ZIPCode As Integer
Public Bedrooms As Short
Public Bathrooms As Single
Public MarketValue As Double
End Class

```

- In the Solution Explorer, right-click RealEstate.vb and click View Code
- In the Class Name combo box, select (RealEstate Events)
- In the Method Name combo box, select Click and change the file as follows:

```

Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Public Class RealEstate

    Private Sub btnNewProperty_Click(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
        Handles btnNewProperty.Click

        Dim Editor As PropertyEditor = New PropertyEditor
        Dim DirInfo As DirectoryInfo = _
            Directory.CreateDirectory("C:\Altair Realtors\Properties")

        Dim RndNumber As Random = New Random
        Dim LeftNumber As Integer = RndNumber.Next(100, 999)
        Dim RightNumber As Integer = RndNumber.Next(100, 999)
        Editor.txtPropertyNumber.Text = LeftNumber.ToString() & "-" _
            & RightNumber.ToString()

        If Editor.ShowDialog() = DialogResult.OK Then
            Dim Prop As SampleProperty = New SampleProperty
            prop.PropertyNumber = Editor.txtPropertyNumber.Text
            prop.PropertyType = Editor.cbxPropertyTypes.Text
            prop.Address = Editor.txtAddress.Text
            prop.City = Editor.txtCity.Text
            prop.State = Editor.cbxStates.Text
            prop.ZIPCode = CInt(Editor.txtZIPCode.Text)
            prop.Bedrooms = CInt(Editor.txtBedrooms.Text)
            prop.Bathrooms = CSng(Editor.txtBathrooms.Text)
            prop.MarketValue = CDbL(Editor.txtMarketValue.Text)

            Dim strFilename As String = DirInfo.FullName & "\" & _
                Editor.txtPropertyNumber.Text & ".prp"
            Dim stmProperty As FileStream = New FileStream(strFilename, _
                FileMode.Create, FileAccess.Write)
            Dim bfmProperty As BinaryFormatter = New BinaryFormatter

            bfmProperty.Serialize(stmProperty, prop)
        End If
    End Sub
End Class

```

- In the Class Name combo box, select btnClose
- In the Method name combo box, select Click and implement the event as follows:

```

Private Sub btnClose_Click(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
    Handles btnClose.Click

    End
End Sub

```

- Execute the application and continuously click the New Property button to create the following properties (let the computer specify the property number):

Property	Address	City	State	ZIP	Beds	Baths	Market
----------	---------	------	-------	-----	------	-------	--------

Type				Code			Value
Single Family	11604 Aldora Avenue	Baltimore	MD	21205	5	3.5	325650
Townhouse	495 Parker House Terrace	Gettysburg	WV	26201	3	2.5	225500
Condominium	5900 24th Street NW #812	Washington	DC	20008	1	1.0	388665
Single Family	6114 Costinha Avenue	Martinsburg	WV	25401	4	3.5	325000
Condominium	10710 Desprello Street #10D	Rockville	MD	20856	1	1.0	528445

10. Close the form and return to your programming environment

De-Serialization

As serialization is the process of storing an object to a medium, the opposite, de-serialization is used to retrieve an object from a stream. To support this, the **BinaryFormatter** class is equipped with the **Deserialize()** method. Like **Serialize()**, the **Deserialize()** method is overloaded with two versions. One of them uses the following syntax:

```
Public Function Deserialize ( _
    serializationStream As Stream _
) As Object
```

This method takes as argument a **Stream**-based object, such as a **FileStream** variable, that indicates where the file is located. The **Deserialize()** method returns an **Object** object. As a goal, you want the **Deserialize()** method to produce the type of object that was saved so you can retrieve the values that the returned object holds. Because the method returns an **Object** value, you must cast the returned value to the type of your class.

Once the **Deserialize()** method has returned the desired object, you can access its values. Here is an example:

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim Vehicle As Car = New Car()

        Vehicle.Make = txtMake.Text
        Vehicle.Model = txtModel.Text
        Vehicle.Year = CInt(txtYear.Text)
        Vehicle.Color = cbxColors.SelectedIndex

        Dim stmCar As FileStream = New FileStream("Car3.car", FileMode.Create)
        Dim bfmCar As BinaryFormatter = New BinaryFormatter()

        bfmCar.Serialize(stmCar, vehicle)
    End Sub

    Private Sub btnRead_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnRead.Click
        Dim stmCar As FileStream = New FileStream("Car3.car", FileMode.Open)
        Dim bnrCar As BinaryReader = New BinaryReader(stmCar)

        Dim bfmCar As BinaryFormatter = New BinaryFormatter()
        Dim Vehicle As Car = CType(bfmCar.Deserialize(stmCar), Car)

        txtMake.Text = Vehicle.Make
        txtModel.Text = Vehicle.Model
        txtYear.Text = Vehicle.Year.ToString()
        cbxColors.SelectedIndex = Vehicle.Color
    End Sub
End Class
```

❖ Practical Learning: De-Serializing an Object

1. In the Class Name combo box, select btnOpen
2. In the Method name combo box, select Click and implement the event as follows:

```
Private Sub btnOpen_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) _
    Handles btnOpen.Click
```

```

Dim DirProperties As DirectoryInfo = _
    New DirectoryInfo("C:\Altair Realtors\Properties")
Dim FileProperties() As FileInfo = DirProperties.GetFiles()

If DirProperties.Exists = True Then
    Dim Found As Boolean = False
    Dim Prop As SampleProperty = Nothing
    Dim file As FileInfo
    For Each file In FileProperties
        Dim stmProperty As FileStream = _
            New FileStream(file.FullName, _
                FileMode.Open, _
                FileAccess.Read)
        Dim bfmProperty As BinaryFormatter = New BinaryFormatter()
        Prop = CType(bfmProperty.Deserialize(stmProperty), _
            SampleProperty)

        If Prop.PropertyNumber = txtPropertyNumber.Text Then
            Found = True
            End If
        Next

    If Found = True Then
        txtPropertyType.Text = Prop.PropertyType
        txtAddress.Text = Prop.Address
        txtCity.Text = Prop.City
        txtState.Text = Prop.State
        txtZIPCode.Text = Prop.ZIPCode
        txtBedrooms.Text = Prop.Bedrooms
        txtBathrooms.Text = FormatNumber(Prop.Bathrooms)
        txtMarketValue.Text = FormatNumber(Prop.MarketValue)
    Else
        MsgBox("There is no property with " & _
            "that number in our database")

        txtPropertyType.Text = "Unknown"
        txtAddress.Text = ""
        txtCity.Text = ""
        txtState.Text = ""
        txtZIPCode.Text = "00000"
        txtBedrooms.Text = "0"
        txtBathrooms.Text = "0.00"
        txtMarketValue.Text = "0.00"
    End If
End If
End Sub

```

3. Execute the application and try opening a previously save property using its number
4. Close the form and return to your programming environment

Details on Serialization

Partial Serialization

In the examples we have used so far, we were saving the whole object. You can make it possible to save only some parts of the class. When creating a class, you can specify what fields would be serialized and which ones would not be. To specify that a member cannot be saved, you can mark it with the **<NonSerialized>** attribute. Here is an example:

```

<Serializable(>> Public Class Carl
    Public Make As String
    Public Model As String

    ' Because the value of a car can change,
    ' there is no reason to save it
    <NonSerialized(>> _
    Public Value As Double
    Public Year As Integer
    Public Color As Integer
End Class

```

After creating the class, you can declare a variable of it and serialize it, using either the binary or the SOAP approach. You can then retrieve the object and its values, using any of the techniques we learned earlier.

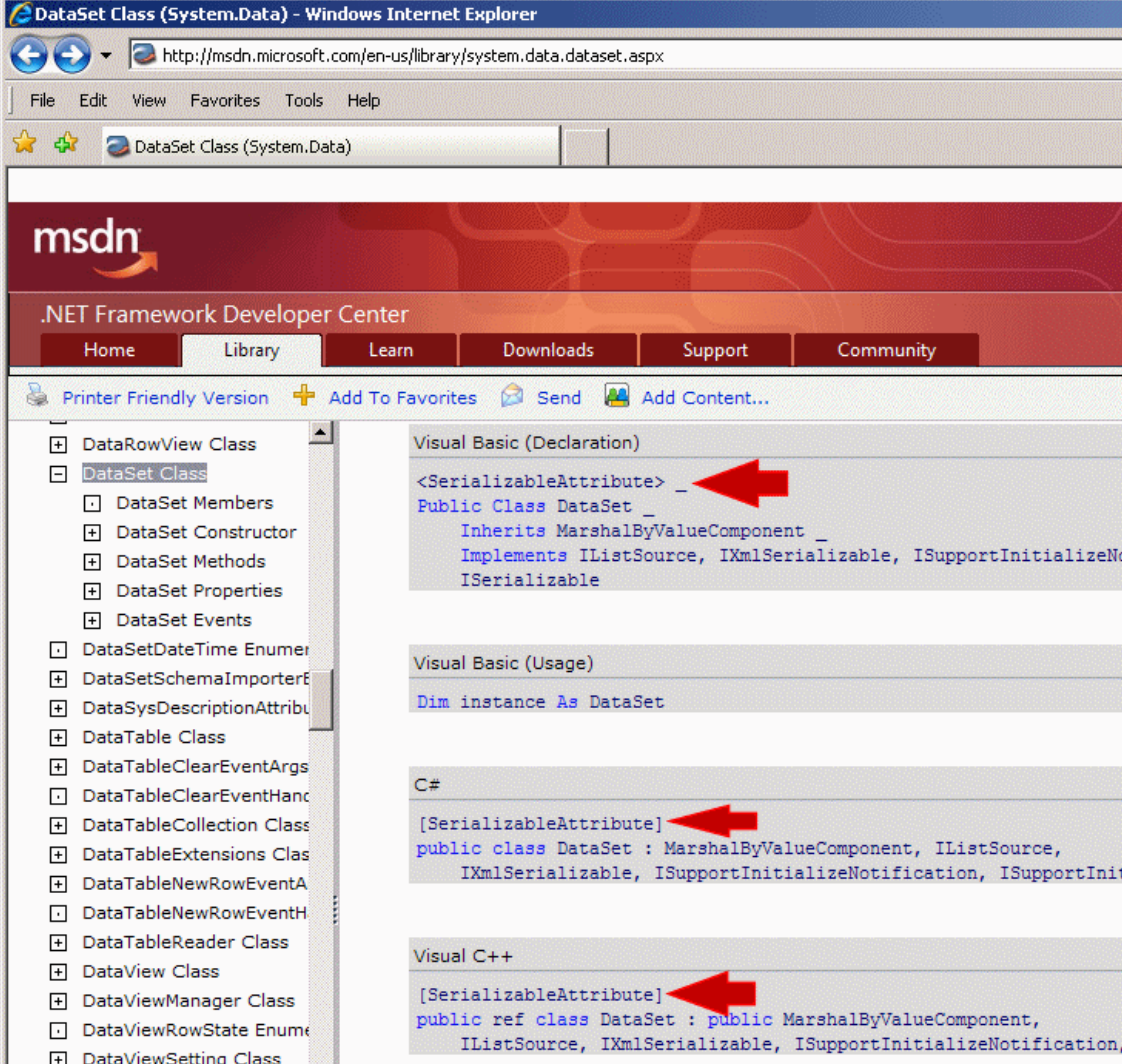
Implementing a Custom Serialized Class

To support serialization, the .NET Framework provides the **ISerializable** interface. You can create a class that implements this interface to customize the serialization process. Even if you plan to use this interface, the class you create must be marked with the **<Serializable>**

attribute.

.NET Built-In Serialized Classes

The .NET Framework is filled with many classes ready for serialization. To know that a class is ready for serialization, when viewing its documentation either in the MSDN web site or in the help documentation, check that it is marked with the **[SerializableAttribute]**. Here is an example of such as class:



The screenshot shows the MSDN website for the DataSet Class (System.Data). The page is titled "DataSet Class (System.Data) - Windows Internet Explorer" and the URL is "http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx". The page features the MSDN logo and the ".NET Framework Developer Center" navigation bar. The left sidebar shows a tree view of the DataSet Class, with "DataSet Class" selected. The main content area displays the class declaration in four languages: Visual Basic (Declaration), Visual Basic (Usage), C#, and Visual C++. Red arrows point to the **<SerializableAttribute>** attribute in the Visual Basic (Declaration) and C# sections, and the **[SerializableAttribute]** attribute in the Visual C++ section.

```

Visual Basic (Declaration)
<SerializableAttribute> _
Public Class DataSet _
    Inherits MarshalByValueComponent _
    Implements IListSource, IXmlSerializable, ISupportInitializeNot
    ISerializable

Visual Basic (Usage)
Dim instance As DataSet

C#
[SerializableAttribute]
public class DataSet : MarshalByValueComponent, IListSource,
    IXmlSerializable, ISupportInitializeNotification, ISupportInit

Visual C++
[SerializableAttribute]
public ref class DataSet : public MarshalByValueComponent,
    IListSource, IXmlSerializable, ISupportInitializeNotification,
  
```

Some of these classes provide the properties and methods to create an object and directly save it. For some other classes, you must first create a class, mark it with the **<Serializable>** attribute, build an object of it, and then pass it to the .NET class.

Unlimited Webspace and Unlimited Bandwidth **FREE**
Domain Registration

⇒ 100 Email Ids of 10 GB **Only Rs. 2000/-**

⇒ All Database FREE ⇒ Full Control Panel

⇒ 5000 Templates / Website Builder ⇒ 200 SEO Tools/Softwares/Weblinks

www.manashosting.com Feedback - Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.



SOAP Serialization

Object Serialization and De-Serialization

Introduction

Consider the following program:

[HTML to PDF with ABCpdf](#)

PDF generation for C#, VB, .NET and ASP. Full HTML / CSS support
www.websupergoo.com

[Luxurious Natural Soap](#)

Manufacturer of quality soaps custom, melt & pour, hotel more
www.sficorp.com

[C#](#)

C, C++, and C# Resources. Find tutorials, tips, and reviews.
www.DevSource.com

[SOAP/Web Services for NSK](#)

Open your NonStop servers to SOA Easy, fast, plus many more benefits
www.nuwave-tech.com



Ads by Google

```
Imports System.IO
```

```
Public Class Exercise
```

```
Private Sub btnWrite_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnWrite.Click
    Dim Make As String = txtMake.Text
    Dim Model As String = txtModel.Text
    Dim Year As Integer = CInt(txtYear.Text)
    Dim CarColor As Integer = cbxColors.SelectedIndex
```

```
    Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Create)
    Dim bnwCar As BinaryWriter = New BinaryWriter(stmCar)
```

```
    Try
```

```
        bnwCar.Write(Make)
        bnwCar.Write(Model)
        bnwCar.Write(Year)
        bnwCar.Write(CarColor)
```

```
    Finally
```

```
        bnwCar.Close()
        stmCar.Close()
```

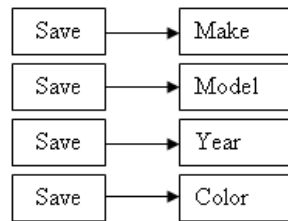
```
    End Try
```

```
End Sub
```

```
End Class
```

Here is an example of running the program:

This is an example of the techniques used in file processing to save [individual](#) data of primitive types:



The values can be retrieved with the following code:

```

Imports System.IO

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim Make As String = txtMake.Text
        Dim Model As String = txtModel.Text
        Dim Year As Integer = CInt(txtYear.Text)
        Dim CarColor As Integer = cbxColors.SelectedIndex

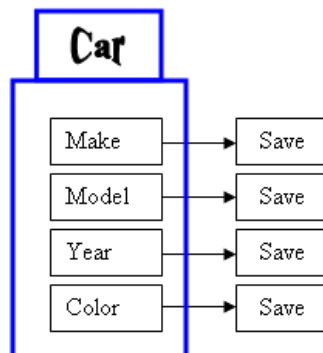
        Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Create)
        Dim bnrCar As BinaryWriter = New BinaryWriter(stmCar)

        Try
            bnrCar.Write(Make)
            bnrCar.Write(Model)
            bnrCar.Write(Year)
            bnrCar.Write(CarColor)
        Finally
            bnrCar.Close()
            stmCar.Close()
        End Try
    End Sub

    Private Sub btnRead_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles btnRead.Click
        Dim stmCar As FileStream = New FileStream("Car1.car", FileMode.Open)
        Dim bnrCar As BinaryReader = New BinaryReader(stmCar)

        Try
            txtMake.Text = bnrCar.ReadString()
            txtModel.Text = bnrCar.ReadString()
            txtYear.Text = bnrCar.ReadUInt32().ToString()
            cbxColors.SelectedIndex = bnrCar.ReadInt32()
        Finally
            bnrCar.Close()
            stmCar.Close()
        End Try
    End Sub
End Class
  
```

In the same way, you can save the individual fields of a class or you can retrieve the individual fields of a car:



Here is an example:

Class: **Car.vb**

```

Public Class Car
    Public Make As String
    Public Model As String
    Public Year As Integer
  
```

```

    Public Color As Integer
End Class

Imports System.IO

Public Class Exercise

    Private Sub btnWrite_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) Handles btnWrite.Click
        Dim Vehicle As Car = New Car()

        Vehicle.Make = txtMake.Text
        Vehicle.Model = txtModel.Text
        Vehicle.Year = CInt(txtYear.Text)
        Vehicle.Color = cbxColors.SelectedIndex

        Dim stmCar As FileStream = New FileStream("Car2.car", FileMode.Create)
        Dim bnwCar As BinaryWriter = New BinaryWriter(stmCar)

        Try
            bnwCar.Write(Vehicle.Make)
            bnwCar.Write(Vehicle.Model)
            bnwCar.Write(Vehicle.Year)
            bnwCar.Write(Vehicle.Color)
        Finally
            bnwCar.Close()
            stmCar.Close()
        End Try
    End Sub

    Private Sub btnRead_Click(ByVal sender As Object, _
                               ByVal e As System.EventArgs) Handles btnRead.Click
        Dim stmCar As FileStream = New FileStream("Car2.car", FileMode.Open)
        Dim bnrCar As BinaryReader = New BinaryReader(stmCar)

        Try
            Dim Vehicle As Car = New Car()

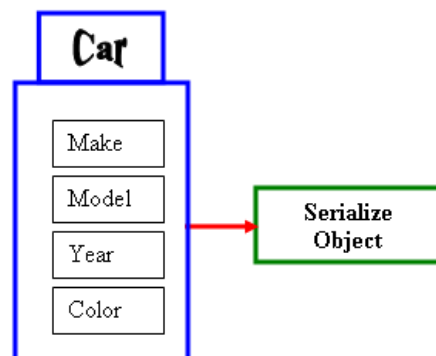
            Vehicle.Make = bnrCar.ReadString()
            Vehicle.Model = bnrCar.ReadString()
            Vehicle.Year = bnrCar.ReadUInt32()
            Vehicle.Color = bnrCar.ReadInt32()

            txtMake.Text = Vehicle.Make
            txtModel.Text = Vehicle.Model
            txtYear.Text = Vehicle.Year
            cbxColors.SelectedIndex = Vehicle.Color
        Finally
            bnrCar.Close()
            stmCar.Close()
        End Try
    End Sub
End Class

```

When it comes to a class, the problem with saving individual fields is that you could forget to save one of the fields. For example, considering a Car class, if you don't save the Make information of a Car object and retrieve or open the saved object on another [computer](#), the receiving user would miss some information and the car cannot be completely identifiable. An alternative is to save the whole Car object.

Object serialization consists of saving a whole object as one instead of its individual fields:



In other words, a variable declared from a class can be saved to a stream and then the saved object can be retrieved later or on another computer. The .NET Framework supports two types of object serialization: binary and SOAP.

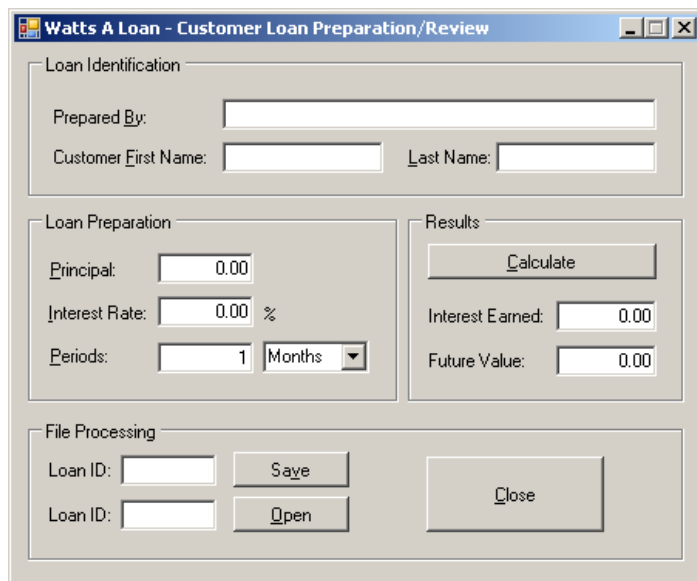
SOAP Serialization

Introduction to SOAP Serialization















The .NET Framework supports a technique of serialization referred to as SOAP (which stands for Simple Object Access Protocol). This technique is related to [XML](#) but, although we haven't studied XML, you don't need to know anything about it to use SOAP serialization.

❖ Practical Learning: Introducing SOAP Serialization

1. Start a new Windows [Application](#) named **Loan1**
2. In the Solution Explorer, right-click Form1.vb and click Rename
3. Type **LoanPreparation.vb** and press Enter
4. [Design](#) the form as followed:



Control	Name	Text	Additional Properties
GroupBox		Loan Identification	
Label	A	Prepared &By:	
TextBox	abl	txtEmployeeName	
Label	A	Customer First Name:	
TextBox	abl	txtCustomerFirstName	
Label	A	Last Name:	
TextBox	abl	txtCustomerLastName	
GroupBox		Loan Preparation	
Label	A	Principal:	
TextBox	abl	0.00	TextAlign: Right
Label	A	Interest Rate:	
TextBox	abl	8.25	TextAlign: Right
Label	A	%	
Label	A	Periods:	
TextBox	abl	1	TextAlign: Text
ComboBox	cbxPeriods	Months	Items: Years Months

				Days
GroupBox			Results	
Button		btnCalculate	Calculate	
Label			Interest Earned:	
TextBox		txtInterestEarned	0.00	TextAlign: Right ReadOnly: True
Label			Amount Earned:	
TextBox		txtFutureValue	0.00	TextAlign: Right ReadOnly: True
GroupBox			File Processing	
Label			Loan ID:	
TextBox		txtSave		
Button		&Save	btnSave	
Label			Loan ID:	
TextBox		txtOpen		
Button		&Open	btnOpen	
Button		btnClose	Close	

- Right-click the form and click View Code
- In the Class Name combo box, select txtCustomerLastName
- In the Method Name combo box, select Leave and implement the event as follows:

```
Private Sub txtCustomerLastName_Leave(ByVal sender As Object, _
                                     ByVal e As System.EventArgs) _
    Handles txtCustomerLastName.Leave
    Dim Initials As String = "00"
    Dim FirstName As String = txtCustomerFirstName.Text
    Dim LastName As String = txtCustomerLastName.Text

    If LastName.Length = 0 Then
        MsgBox("You must enter a last name")
        txtCustomerLastName.Focus()
        Exit Sub
    End If

    If FirstName.Length = 0 Then
        Initials = LastName.Substring(0, 1) & "1"
    Else
        Initials = FirstName.Substring(0, 1) & LastName.Substring(0, 1)
    End If

    txtSave.Text = Initials
End Sub
```

- In the Class Name combo box, select btnCalculate
- In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnCalculate_Click(ByVal sender As Object, _
                               ByVal e As System.EventArgs) _
    Handles btnCalculate.Click
    Dim Principal As Double
    Dim InterestRate As Double
    Dim InterestEarned As Double
    Dim FutureValue As Double
    Dim Periods As Double

    ' Retrieve the value of the principal
    Try
        Principal = Cdbl(txtPrincipal.Text)
    Catch ex As FormatException
        MsgBox("The value you entered for the principal " & _
              "is not valid.\nPlease try again")
    End Try
```

Ads by Google

**Soap Testing**

Simple & easy to use
SOAP load testing tool.
Download Now!

www.Paessler.com/soap-load-tester

String Lubricant

Buy All Natural Guitar
Polish & Accessories. Free
Shipping!

www.ZZGuitarWorks.com

Hotel & Spa Accessories

Bali's Largest & Finest
Range Custom orders
welcome

www.sb-he.com

Convert Text to XML

Convert any Text Format
to XML Java and .NET
Converter Components

www.xmlconverters.com/Convert-Ti

ECO for Visual Studio

Much more than just an
ORM Download free
version today

capableobjects.com

```

' Retrieve the interest rate
Try
    InterestRate = CDbI(txtInterestRate.Text) / 100
Catch ex As FormatException
    MsgBox("The value you entered for the interest " & _
        "rate is not valid\nPlease try again")
End Try

' Get the number of periods
Try
    If cbxPeriods.SelectedIndex = 0 Then ' Years
        Periods = CDbI(txtPeriods.Text)
    ElseIf cbxPeriods.SelectedIndex = 1 Then ' Months
        Periods = CDbI(txtPeriods.Text) / 12
    Else ' if cbxPeriods.SelectedIndex = 2) Days
        Periods = CDbI(txtPeriods.Text) / 360
    End If

Catch ex As FormatException
    MsgBox("The value you entered for the number " & _
        "of periods is not valid\nPlease try again")
End Try

Dim InterestRatePeriods As Double = InterestRate * Periods
Dim InterestPlus1 As Double = InterestRatePeriods + 1
FutureValue = Principal * InterestPlus1
InterestEarned = FutureValue - Principal

txtInterestEarned.Text = FormatNumber(InterestEarned)
txtFutureValue.Text = FormatNumber(FutureValue)
End Sub

```

10. Execute the application to make sure it is fine
11. After using it, close the form and return to your [programming environment](#)

Serialization With SOAP

To serialize an object using SOAP, you follow the same steps we reviewed for the binary serialization with one addition: you must add a certain reference.

When creating the class whose objects would be serialized, mark it with the **<Serializable>** attribute. Here is an example:

```

<Serializable()> Public Class Car
    Public Make As String
    Public Model As String
    Public Year As Integer
    Public Color As Integer
End Class

```

To support SOAP serialization, the .NET Framework provides the **SoapFormatter** class. This class is defined in the **System.Runtime.Serialization.Formatters.Soap** namespace that is part of the **System.Runtime.Serialization.Formatters.Soap.dll** assembly. In order to use The **SoapFormatter** class, you must reference this assembly. Then, you can create an object and initialize it as you see fit. Before saving it, as always, create a **Stream**-based object that would indicate the name (and location) of the file and the type of action to perform. Then, declare a **SoapFormatter** dim as iable using its default constructor. To actually save the object, call the **Serialize()** method of this class. This method uses the same syntax as that of the **BinaryFormatter** class: it takes two arguments. The first is a Stream-based object. The second is the object that needs to be serialized.

❖ Practical Learning: Serializing With SOAP

1. To create a new class, on the main menu, click Project -> Add Class...
2. Set the Name to **LoanInformation** and click Add
3. Change the file as follows:

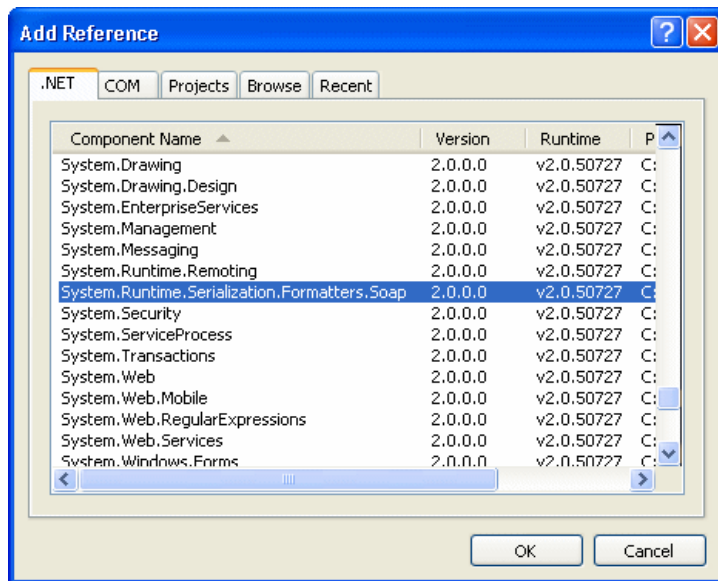
```

<Serializable()> Public Class LoanInformation
    Public EmployeeName As String
    Public CustomerFirstName As String
    Public CustomerLastName As String
    Public Principal As Double
    Public InterestRate As Double
    Public Periods As Double
    Public PeriodType As Integer
End Class

```

4. To add SOAP support to your project, on the main menu, click Project -> Add Reference...

5. In the Add Reference dialog box and in the .NET tab, scroll down and select **System.Runtime.Serialization.Formatters.Soap**:



6. Click OK
7. Access the LoanPreparation.vb code file
8. In the top section of the form, type the following:
- ```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap

Public Class LoanPreparation
```
9. In the Class Name combo box, select btnSave
10. In the Method Name combo box, select Click and implement the event as follows:

```
Private Sub btnSave_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) _
 Handles btnSave.Click
 If txtSave.Text.Length = 0 Then
 MsgBox("Please enter the customer " & _
 "initials or a name for the loan")
 txtSave.Focus()
 Exit Sub
 End If

 Dim infLoan As LoanInformation = New LoanInformation

 infLoan.EmployeeName = txtEmployeeName.Text
 infLoan.CustomerFirstName = txtCustomerFirstName.Text
 infLoan.CustomerLastName = txtCustomerLastName.Text
 infLoan.Principal = CDb1(txtPrincipal.Text)
 infLoan.InterestRate = CDb1(txtInterestRate.Text)
 infLoan.Periods = CDb1(txtPeriods.Text)
 infLoan.PeriodType = cbxPeriods.SelectedIndex

 Dim stmLoan As FileStream = New FileStream(txtSave.Text, _
 FileMode.Create, _
 FileAccess.Write)

 Dim sfmLoan As SoapFormatter = New SoapFormatter

 Try
 sfmLoan.Serialize(stmLoan, infLoan)

 txtEmployeeName.Text = ""
 txtCustomerFirstName.Text = ""
 txtCustomerLastName.Text = ""
 txtPrincipal.Text = "0.00"
 txtInterestRate.Text = "0.00"
 txtPeriods.Text = "0"
 cbxPeriods.SelectedIndex = 0
 txtFutureValue.Text = "0.00"
 txtInterestEarned.Text = "0.00"
 txtSave.Text = ""
 txtOpen.Text = ""
 End Try
End Sub
```

```

 txtEmployeeName.Focus()
 Finally
 stmLoan.Close()
 End Try
End Sub

```

11. Press Ctrl + F5 to execute the application
12. Create, calculate, and save a few loans

13. Close the form and return to your programming environment

## De-Serialization With SOAP

De-serialization in soap is performed exactly as done for the binary de-serialization. To support it, the **SoapFormatter** class is equipped with the **Deserialize()** method. This method uses the same syntax as its equivalent of the **BinaryFormatter** class. The approach to use it is also the same.

### ❖ Practical Learning: Deserializing With SOAP

1. In the Class Name combo box, select btnOpen
2. In the Method Name combo box, select Click
3. To deserialize, implement the event as follows:

```

Private Sub btnOpen_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) _
 Handles btnOpen.Click
 If txtOpen.Text.Length = 0 Then

```



```

 MsgBox("Please enter a customer's initials or " & _
 "a name given to a previous loan preparation")
 txtOpen.Focus()
 Exit Sub
End If

Try
 Dim LoanInfo As LoanInformation = New LoanInformation
 Dim LoanStream As FileStream = New FileStream(txtOpen.Text, _
 FileMode.Open, _
 FileAccess.ReadWrite)
 Dim LoanFormatter As SoapFormatter = New SoapFormatter

 Try
 ' Open the file and store its values
 ' in the LoanInformation variable
 LoanInfo = CType(LoanFormatter.Deserialize(LoanStream), _
 LoanInformation)

 ' Retrieve each value and put it in its corresponding control
 txtEmployeeName.Text = LoanInfo.EmployeeName
 txtCustomerFirstName.Text = LoanInfo.CustomerFirstName
 txtCustomerLastName.Text = LoanInfo.CustomerLastName
 txtPrincipal.Text = LoanInfo.Principal.ToString("F")
 txtInterestRate.Text = LoanInfo.InterestRate.ToString("F")
 txtPeriods.Text = LoanInfo.Periods.ToString()
 cbxPeriods.SelectedIndex = LoanInfo.PeriodType

 ' Since we didn't save the calculated values,
 ' call the Click event of the Calculate button
 btnCalculate_Click(sender, e)
 Finally
 LoanStream.Close()
 End Try
Catch ex As FileNotFoundException
 MsgBox("There is no file with that name")
End Try
End Sub

```

4. In the Class Name combo box, select btnOpen
5. In the Method Name combo box, select Click and implement the event as follows:

```

Private Sub btnClose_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) _
 Handles btnClose.Click
 End
End Sub

```

6. Press Ctrl + F5 to execute the application
7. Enter the initials of a previously created loan and click Open
8. Close the form and return to your programming environment

## Details on Serialization

### Partial Serialization

In the examples we have used so far, we were saving the whole object. You can make it possible to save only some parts of the class. When creating a class, you can specify what fields would be serialized and which ones would not be. To specify that a member cannot be saved, you can mark it with the **<NonSerialized>** attribute. Here is an example:

```

<Serializable()> Public Class Carl
 Public Make As String
 Public Model As String

 ' Because the value of a car can change,
 ' there is no reason to save it
 <NonSerialized()> _
 Public Value As Double
 Public Year As Integer
 Public Color As Integer
End Class

```

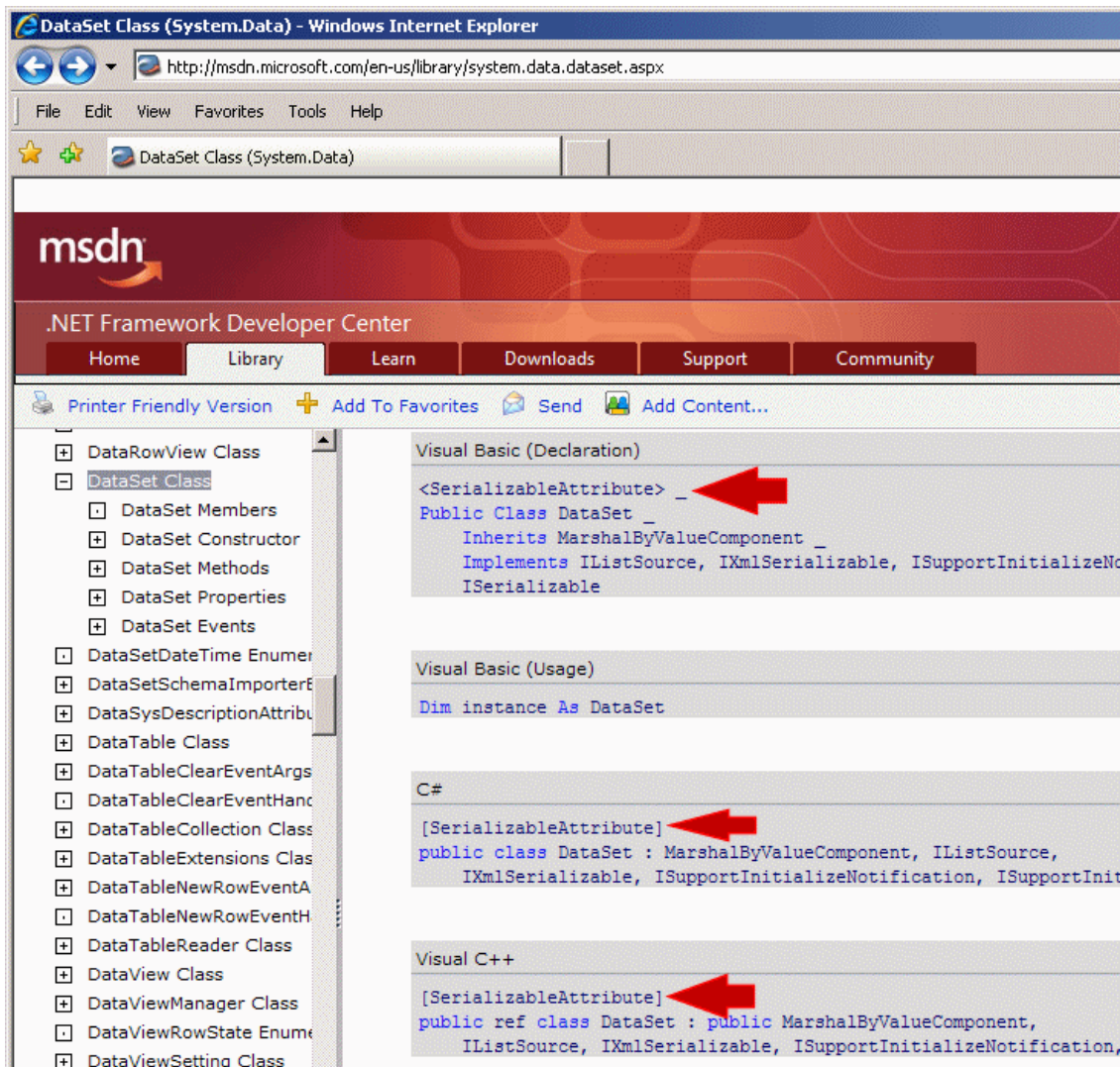
After creating the class, you can declare a variable of it and serialize it, using either the binary or the SOAP approach. You can then retrieve the object and its values, using any of the techniques we learned earlier.

### Implementing a Custom Serialized Class

To support serialization, the .NET Framework provides the **ISerializable** interface. You can create a class that implements this interface to customize the serialization process. Even if you plan to use this interface, the class you create must be marked with the **<SerializableAttribute>** attribute.

## .NET Built-In Serialized Classes

The .NET Framework is filled with many classes ready for serialization. To know that a class is ready for serialization, when viewing its documentation either in the MSDN web site or in the help documentation, check that it is marked with the **[SerializableAttribute]**. Here is an example of such as class:



The screenshot shows the MSDN website for the DataSet Class (System.Data). The page displays the class declaration in Visual Basic, Visual Basic (Usage), C#, and Visual C++.

**Visual Basic (Declaration)**

```
<SerializableAttribute> _
Public Class DataSet _
 Inherits MarshalByValueComponent _
 Implements IListSource, IXmlSerializable, ISupportInitializeNot
 ISerializable
```

**Visual Basic (Usage)**

```
Dim instance As DataSet
```

**C#**

```
[SerializableAttribute]
public class DataSet : MarshalByValueComponent, IListSource,
 IXmlSerializable, ISupportInitializeNotification, ISupportInit
```

**Visual C++**

```
[SerializableAttribute]
public ref class DataSet : public MarshalByValueComponent,
 IListSource, IXmlSerializable, ISupportInitializeNotification,
```

Some of these classes provide the properties and methods to create an object and directly save it. For some other classes, you must first create a class, mark it with the **<SerializableAttribute>** attribute, build an object of it, and then pass it to the .NET class.

**ASP.NET OLAP Control** - Want give a Web app OLAP functions?  
Easy with RadarCube! DBMS or MSAS. [www.radar-soft.com](http://www.radar-soft.com)

Ads by Google

[Home](#)

Copyright © 2008 FunctionX, Inc.