Windows-based Applications with

Microsoft

Visual C# .NET and Microsoft Visual Studio .Net

Version 3.0.0

Developing and Implementing Windows-based Applications with

Microsoft Visual C# .NET and Microsoft Visual Studio .Net

Version 3.0.0

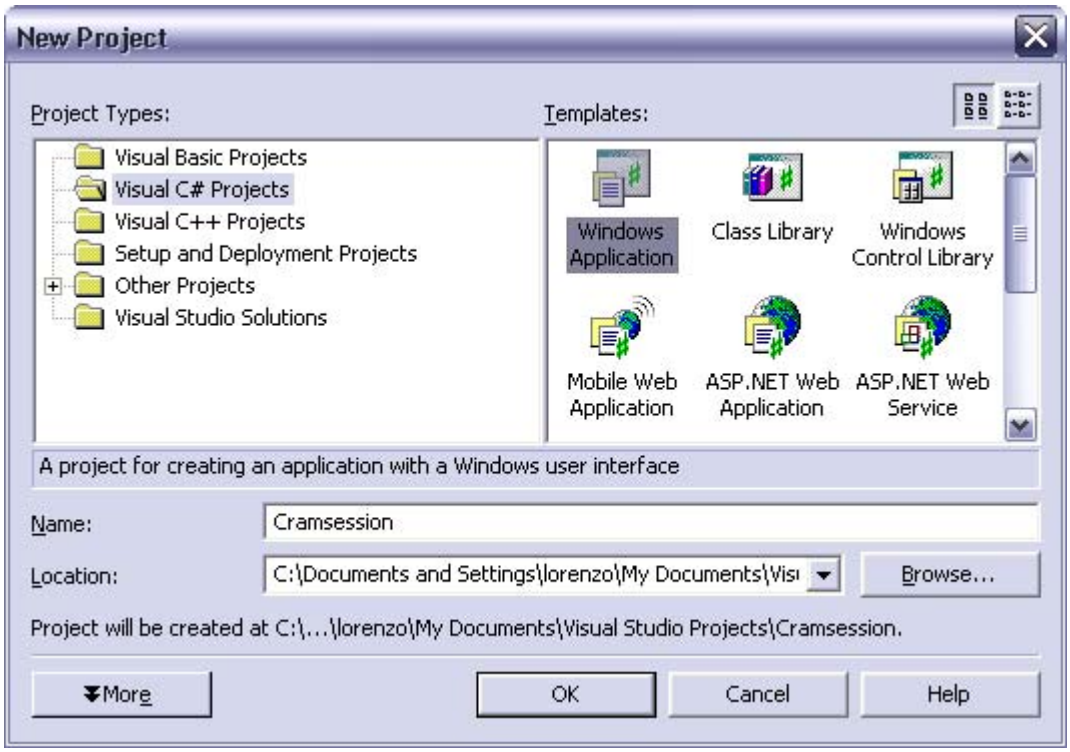
Abstract:

This study guide will help you prepare for Microsoft exam 70-316, Developing and Implementing Windows-based Applications with Microsoft Visual C# .NET and Microsoft Visual Studio .Net. Exam topics include: Creating User Services; Creating and Managing Components and .NET Assemblies; Consuming and Manipulating Data; Testing and Debugging; Deploying a Windows-based Application; Maintaining and Supporting a Windows-based Application; and Configuring and Securing a Windows-based Application.

Contents:

Creating User Services

To start creating a new Windows Application written in C#, you must select the **New Project** button or menu command. The New Project Wizard will appear:



First choose Visual C# Projects and then choose Windows Application.

You can then specify the name and the location of the project.

Create a Windows Form by using the Windows Forms Designer

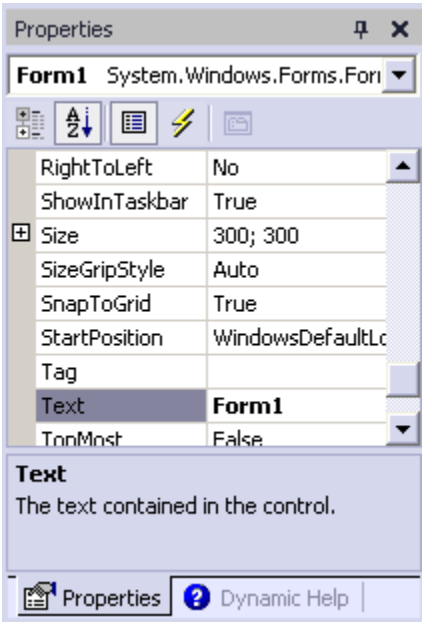
The first form of a project is created with the project.

To create a new form, use the **Add -> Add Windows Forms** command in the Solution Explorer.

Add and set properties on a Windows Form

To set the properties of a Form (and of any other Control) use the Properties Window. It's possible to group similar properties, and to sort them alphabetically.

The Properties Window is used to show not only properties but also the events that the form receives.



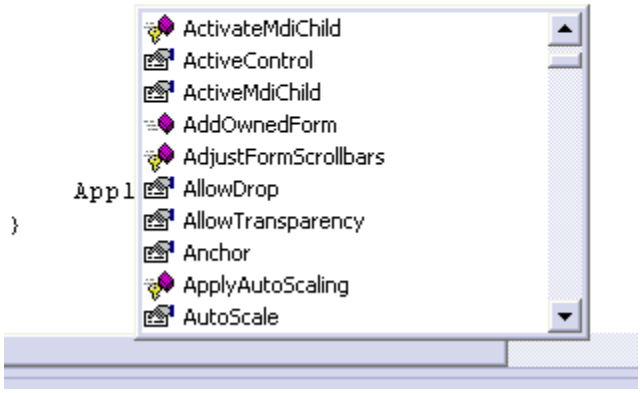
The most used properties of a Form are:

(Name)	Name of the Form
AcceptButton	Specifies which button is clicked when the user presses the ENTER key
AllowDrop	Specifies if the Form accepts drag-n-drop messages
AutoScale	Specifies if the Form will scale depending on the font
AutoScroll	Enables the scroll bars if controls are placed outside the visible area of the Form
BackColor and ForeColor	Specifies the colors of the Form
BackgroundImage	Specifies an image used as the background of the Form
CancelButton	Specifies which button is clicked when the user press the ESC key
ControlBox, MaximizeBox, MinimizeBox, HelpButton	ControlBox enables the control box in the caption bar. The control box can contain the Minimize, the Maximize, the Help and the Close button. Each single button must be enabled independently.
FormBorderStyle	Specifies how the border of the Form is displayed (None, Sizeable, Fixed, etc.)
IsMDIContainer	Specifies if the Form is an MDI Container or not
Language	Specifies the default language of the Form
Localizable	Enables the generation of localizable code in the Form
Locked	Enables the ability to move and resize controls
Opacity	Enables the transparency of the Form (0% transparent, 100% opaque)
ShowInTaskbar	Enables the presence of the Form in the Taskbar
Size (Width, Height)	Specifies the initial dimensions of the Form
StartPosition	Is the initial position of the Form on the screen
Text	Is the text in the caption bar
TopMost	Specifies if the Window is displayed over other non top-most windows also when

	inactive
WindowState	Is the initial state of the Window (Minimized, Maximized, Normal)

It's also possible to set every property of a form with the code:

```
static void Main()
{
    Form1 myForm = new Form1();

    myForm.AutoScale = true;
    Appl
}
```



Create a Windows Form by using visual inheritance

Visual Inheritance is used to inherit a form from an existing one, an efficient way to reuse the implementation of a form without rewriting it. The inherited form can then modify some of the properties, the events and the controls of the original form, without worrying about the other implementation details.

Visual Inheritance can be implemented in two ways:

1. By adding a normal form to the Project, adding the reference to the original form in the new form and then modifying the definition of the new form:

   ```
   Public class NewForm : OldFormNamespace.OldForm
   ```

   In this way the NewForm inherits from the OldForm like any other inherited class.

2. By using the **Add Inherited Form** command in the Project Menu of the Visual Studio.NET

Like any other inheritance between classes, it's possible to specify some Access Modifiers:

- **Public**: it's possible to set all the properties of every control of the base form
- **Private**: in this case the properties cannot be modified and are disabled
- **Protected**: only the inherited form can modify the properties of the base form

Build graphical interface elements by using the System.Drawing namespace

The System.Drawing namespace contains all the classes needed to draw on a Form, such as Bitmap, Brush, Font, Graphics, Icon, Image, Pen, StringFormat and many others. It also contains a set of structures like Color, Point, Rectangle, Size, etc…, some enum and some delegates.

The Graphics class contains the methods that allow drawing on a GDI+ drawing surface.

A Graphics class object can be created in a Form or in a Control or it can be obtained from an HDC, from an HWND, from an Image or from other .NET classes like PaintEventArgs (that is passed to the OnPaint event handler).

Graphics use methods to draw lines, rectangles, arcs, curves, Bezier splines, texts, ellipses, images, and so on.

Remember that GDI+ is a stateless graphics system. Every graphic property (like Pen, Brush or Font) must be passed explicitly to every method call.

For example, the method called DrawString, which is used to print a string, has various overloaded versions, but all of them have the first three parameters in common:

```
DrawString(string str, Font font, Brush brush, etc...);
```

where str is the string to be printed, and font and brush defines how the string is printed.

For example, a call to DrawString can be:

```
DrawString("C# Cramsession", this.Font, Brushes.Black, etc...);
```

where this refers to the current form, and Brushes is a class that contains the definitions of the 141 default brushes (similar to the Pens class that contains all the 141 default pens, or to the Color structures that contains all the default 141 colors), from AliceBlue to the YellowGreen color.

Remember also that there is a class called SystemColors that contains all the 26 system defined colors. This is useful to adapt the colors of the application to the color settings of the current system (defined in the Control Panel).

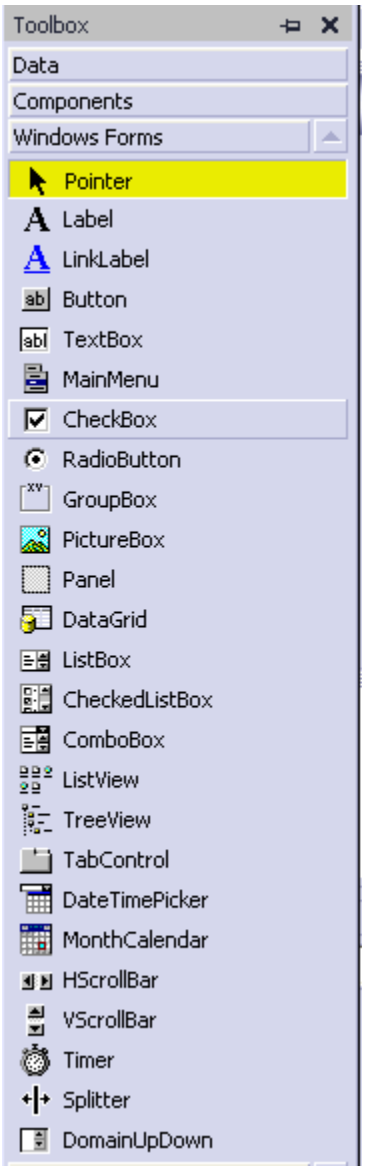
Add controls to a Windows Form

Controls are objects (that derive from Windows.Forms.Control) like buttons, labels, combo-boxes, text boxes, etc... Controls can be created in two ways. The first way is uses the Windows Forms controls in the Visual Studio.NET Toolbox.

It's possible to double click on one or more controls and they will go on the top left area of the form. You can then move every control into position, and change its size.

You can also click on one control in the Toolbox and then draw it on the form with the right size and position.

Here is a sample of the available controls:



The **Format** menu is used to modify the position, the size and the locking of the controls on the form. The available options are:

Align	Aligns a group of controls to the primary control
Make Same Size	Resizes a group of controls
Horizontal Spacing	Sets the horizontal spacing of a group of controls
Vertical Spacing	Sets the vertical spacing of a group of controls
Center in Form	Centers a form horizontally or vertically
Order	Sets the Z-order of a control
Lock Controls	Locks all controls of the form

Load controls dynamically

The second way to create a control is by writing code directly:

```
private System.Windows.Forms.Button button1;
...
button1 = new System.Windows.Forms.Button();


button1.Name = "button1";
button1.TabIndex = 0;
button1.Text = "button1";
button1.Location = new System.Drawing.Point(8, 8);

...

myForm.Controls.Add(button1);
```

This method allows adding controls to a Form at runtime, depending on the user's choice. The control must be declared and created, and then its properties can be set and it can be added to the Controls collection of the Form (or of another control called Container).

Set properties on controls

Properties on controls can be set using the Properties Windows or by writing a line of code like in the previous paragraph.

Write code to handle control events and add the code to a control

Look ahead to the Create, implement, and handle events section.

Instantiate and invoke an ActiveX® control

Windows Forms can host only Windows Forms controls (derived from System.Windows.Forms.Control). ActiveX controls are based on COM, so they need a wrapper that allows hosting them on a Windows Form.

The wrapper can be generated with Visual Studio.NET or with AXIMP.EXE from the command line.

The generated wrapper derives from System.Windows.Forms.AxHost and acts like a Windows Forms control, but it contains an instance of the ActiveX control. It knows how to communicate with the ActiveX control and expose all of its properties, methods and events to the .NET world.

Configure control licensing

The .NET Framework has a licensing model identical for all types of components and controls that is fully compatible with the licensing rules used for ActiveX controls.

Licensing allows authors to protect intellectual property by checking, at design time, that the control is used legally. If this happens, the application gets a run-time license that can be freely distributed.

There are many licensing models, but the simplest requires a set of steps:

1. Include the System.ComponentModel namespace
2. Apply a LicenseProviderAttribute to the class that implements the control
3. Declare a License object
4. Call the **Validate** or **IsValid** methods of the LicenseManager in the constructor of the control to fill the License Object with a valid license

5. Call **Dispose** on any granted license in the finalizer of the class or before the finalizer is called

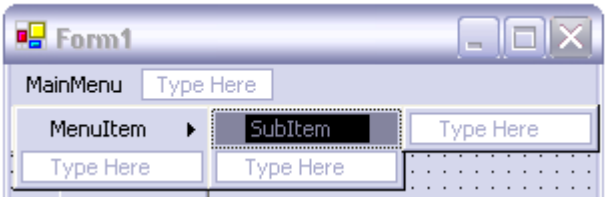
6. Create a text .LIC file (with the correct information) and save it to the assembly folder

Here is an example:

```
using System;
using System.ComponentModel;
using System.Windows.Forms;

[LicenseProvider(typeof(LicFileLicenseProvider))]
public class MyControl : Control {
  private License license = null;

  public MyControl () {
    license = LicenseManager.Validate(typeof(MyControl), this);
  }

  protected override void Dispose(bool disposing) {
    if (disposing) {
      if (license != null) {
        license.Dispose();
        license = null;
      }
    }
    base.Dispose(disposing);
  }

  ~MyControl() {
    Dispose();
  }
}
```

Create menus and menu items

To add a menu to a Form use the MainMenu control on the Toolbox window is used. The menu must be created and then the Menu property of the form must be set to point the to menu.



More than one MainMenus can be created and they can be assigned to the Menu-property of the Form in runtime, depending on the current state of the application.

To create a ContextMenu (the menu that is usually displayed with a mouse right-click on a Form or on a Control) the ContextMenu control on the Toolbox window is used.

The ContextMenu can be assigned to the Form or to a specific control by setting the ContextMenu property.

Like every other Windows.Forms element, the MainMenu and ContextMenu can be created with code.

To add an item to a menu at runtime, a new MenuItem object must be created, and then the Text property can be set. To assign the new MenuItem to an existing menu, use the Add method in the MenuItems collection of the menu.

To add a shortcut to a MenuItem, prefix the letter with the "&" character in the Text property; i.e., myMenuItem.Text = "&Text".

To add an image to a MenuItem, subclass the MenuItem class, override the OnPaint method to draw the image on the left of the text, and use the new class instead of the original one.

Implement navigation for the user interface (UI)

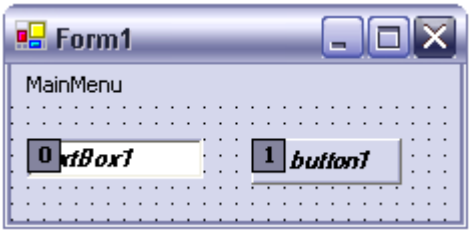
Configure the order of tabs

MSDN- Setting the Tab Order in Windows Forms
The TabStop property of every control enables or disables the navigation using the Tab key.

Use the Tab Order item in the View menu to set the order of tabs.

The number displayed in the top left corner of the control is the order, from 0 to the number of controls.

The other way is to set the TabIndex of every control.



Validate user input

The best way to validate user input is to check the status of a control before it loses focus. The **Validating** event is raised before a control loses focus and only if the **CausesValidation** property of the control, which will receive focus, is set to True.

The Validating event can be used to validate the data in the control, to prevent that a control loses focus or both.

If the CausesValidation property of a control is set to False, it will receive focus without firing the Validating event on the control that has lost the focus. This can be useful, for example, with the Help button. By default, the CausesValidation property is set to True for every control.

To add the Validating event to a control, simply go to the properties window and double click the Validating event. The Visual Studio.NET will add the default validation handler:

```
private void textBox1_Validating(object sender,
System.ComponentModel.CancelEventArgs e)
      {

      }
```

Insert the validation routine in the handler and set **e.Cancel = true** if the validation has failed.

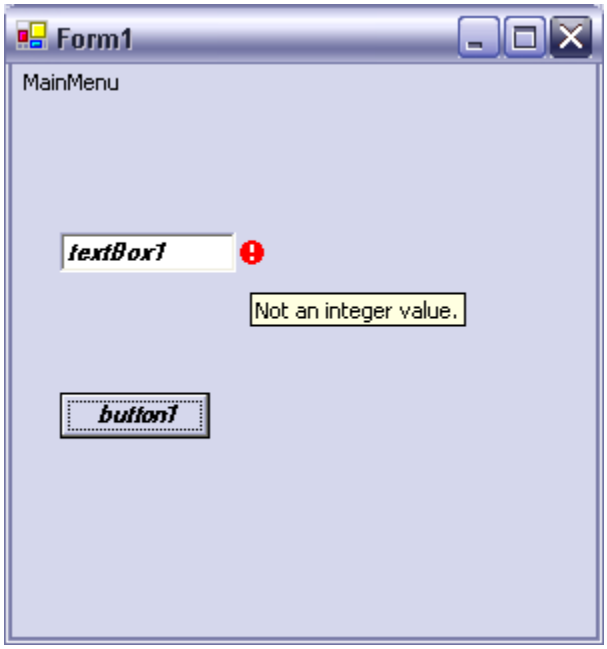
To display the result of a validation, the ErrorProvider control can be used.

Simply create an ErrorProvider control using the Toolbox window, and in the Validating event of a control, call:

```
errorProvider1.SetError(textBox1, "Error message...");
```

If there is an error a red exclamation point is displayed, and when the mouse is over it, the tooltip will display the associated message.

To change the icon displayed by the ErrorProvider, simply create a new icon and set it to the icon property of the ErrorProvider.



Implement error handling in the UI

Create and implement custom error handlers. Raise and handle errors

In traditional languages, errors are generally reported by every single code statement as error codes, and must be handled with if statements. This approach tends to make the code unreadable.

C#, like many other object-oriented languages, allows handling of errors using exceptions.

Put the code inside a try…catch…finally block:

```
try {
      code that can generate an exception
}
catch ( exceptionClass excObject )
{
      code that handles the exception
}
finally
{
      code that is always executed (clean-up code)
}
```

There can be more than one catch block, from the most detailed exception to the most general one.

Catch and "finally" blocks are optional, but at least one of them is needed.

Exceptions are objects derived from System.Exception. It's possible to define custom exceptions (that derive from System.ApplicationException) or to use system exceptions (that derive from System.SystemException):

AppDomainUnloadedException, ArgumentException, ArithmeticException, ArrayTypeMismatchException, BadImageFormatException, CannotUnloadAppDomainException, ComponentModel.Design.Serialization.CodeDomSerializerException, ComponentModel.LicenseException, ComponentModel.WarningException, Configuration.ConfigurationException, Configuration.Install.InstallException, ContextMarshalException, Data.DataException, Data.DBConcurrencyException, Data.SqlClient.SqlException, Data.SqlTypes.SqlTypeException, Drawing.Printing.InvalidPrinterException, EnterpriseServices.RegistrationException, EnterpriseServices.ServicedComponentException, ExecutionEngineException, FormatException, IndexOutOfRangeException, InvalidCastException, InvalidOperationException, InvalidProgramException, IO.InternalBufferOverflowException, IO.IOException, Management.ManagementException, MemberAccessException, MulticastNotSupportedException, NotImplementedException, NotSupportedException, NullReferenceException, OutOfMemoryException, RankException, Reflection.AmbiguousMatchException, Reflection.ReflectionTypeLoadException, Resources.MissingManifestResourceException, Runtime.InteropServices.ExternalException, Runtime.InteropServices.InvalidComObjectException, Runtime.InteropServices.InvalidOleVariantTypeException, Runtime.InteropServices.MarshalDirectiveException, Runtime.InteropServices.SafeArrayRankMismatchException, Runtime.InteropServices.SafeArrayTypeMismatchException, Runtime.Remoting.RemotingException, Runtime.Remoting.ServerException, Runtime.Serialization.SerializationException, Security.Cryptography.CryptographicException, Security.Policy.PolicyException, Security.SecurityException, Security.VerificationException, Security.XmlSyntaxException, ServiceProcess.TimeoutException, StackOverflowException, Threading.SynchronizationLockException, Threading.ThreadAbortException, Threading.ThreadInterruptedException, Threading.ThreadStateException, TypeInitializationException, TypeLoadException, TypeUnloadedException, UnauthorizedAccessException,

Web.Services.Protocols.SoapException, Xml.Schema.XmlSchemaException, Xml.XmlException, Xml.XPath.XPathException, Xml.Xsl.XsltException

If the catch block can't handle the exception that was caught, it can send it to the next handler, using the statement:

```
throw excObject;
```

where excObject is the object that contains the handled exception.

The throw statement is also used to raise an exception inside a class, a method or a piece of code surrounded by try…catch…finally, with the following syntax:

```
throw new exceptionClass(...);
```

Implement online user assistance

There are three ways to provide assistance to the user:

- ToolTips
- Help Menu
- Context-sensitive help

Using ToolTips is very simple. Simply add a ToolTip control to every Form that needs it and then set the "**ToolTip on …**" property on each control of the Form.

To implement the Help Menu simply add the Main Menu to the Form, add the menu item to it and then write this statement in the menu click event handler:

```
Help.ShowHelp( parentOfTheHelpBox, HelpProvider.HelpNamespace );
Where
```

- parentOfTheHelpBox can be this in case of the Main Form or the pointer to another form.

- HelpProvider. HelpNamespace is the file name of the Help file, in the form of "c:\file…" or "file:///..." or http://...

To implement Context-sensitive help in an application, the HelpProvider control, from the Toolbox window, is used.

The HelpNamespace property of the HelpProvider sets the filename or the URL of the help file.

If the HelpNamespace is not set, when the F1 key is pressed, the HelpString property of the control is displayed.

If the HelpNamespace is set when the F1 key is pressed the help file is displayed.

The HelpKeyword sets the search anchor tag ("A NAME='searchString' ") that will be searched in the HTML Help File.

Display and update data

Bind data to the UI

It's possible to bind data to Windows Forms controls not only from traditional data sources (like ADO.NET DataSets), but also to almost any structure that contains data like arrays, collections, properties or other controls.

Any object that derives from the Control class has a ControlBindingsCollection (exposed in the DataBindings property) that can contain Binding objects.

The Binding class is used to maintain a simple binding between a property of the Control (for example Text for a TextBox) and a property of an object.

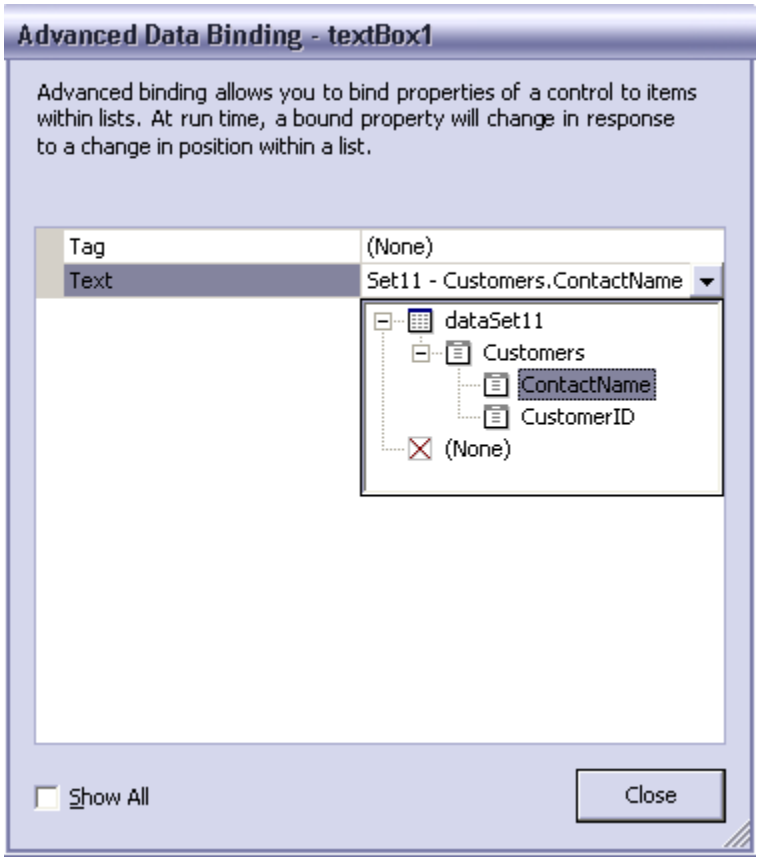
It's possible to bind a data source to a property by using the Property Window or programmatically, for example:

```
txtCustName.DataBindings.Add("Text", dsMyDataSet.Customers, "Name");
```

where the first parameter is the destination property, the second parameter is the source object and the third parameter is the source field in the source object.

When there is a change in the destination property, the change is also done on the source object. If the object is a DataSet, the data in the DB are not modified. The DataSet must be synchronized with the DB.

It's possible to set bindings by selecting Advanced Data Binding properties from the properties window.



Transform and filter data

There are situations where data retrieved from a data source does not match with the data type of the destination control.

It's possible to handle two events, **Format** and **Parse** that allow transforming the data before sending them to the control and vice versa.

These events also allow creating custom formats for displaying data.

The Format event is fired when the property is bound, when the Position changes or when the data is changed (sorted or filtered).

The Parse event is fired after the Validated event, when the Position changes or when the EndCurrentEdit method is called.

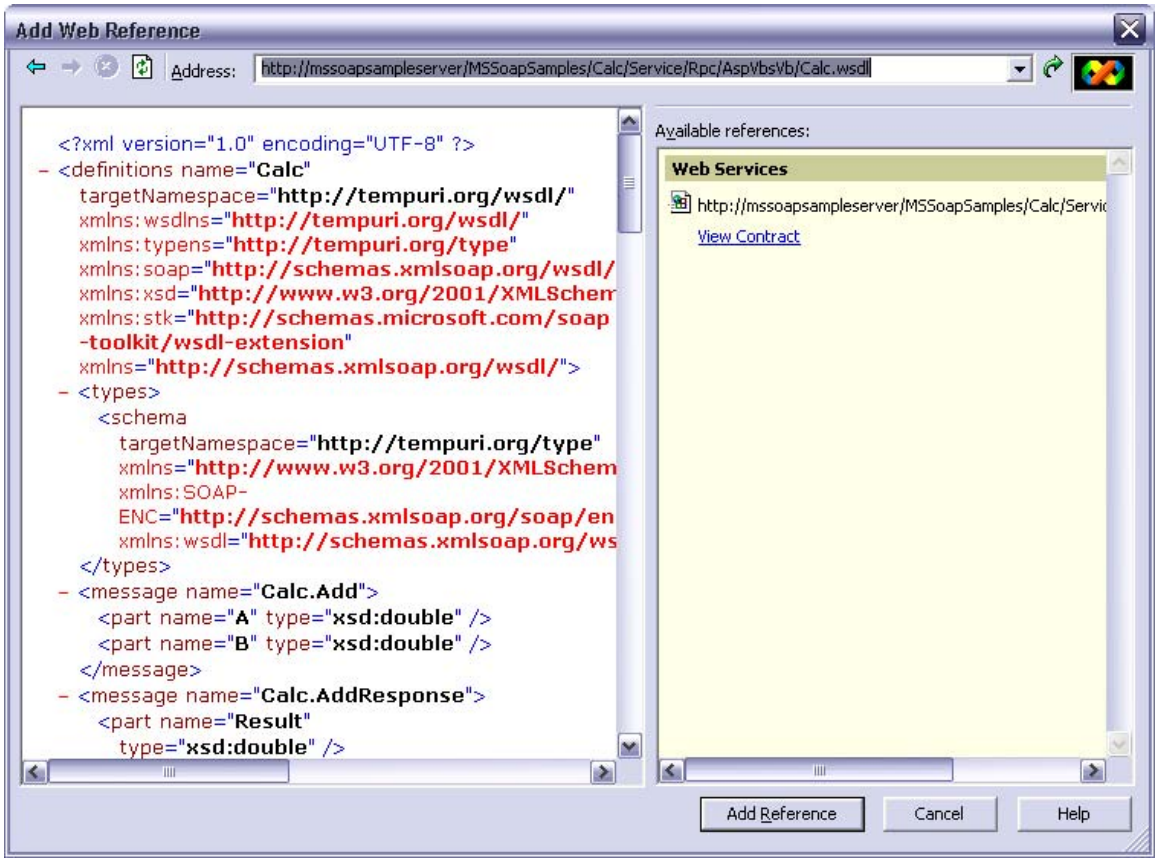
It's possible to filter data in a DataSet using the Select statement or a DataView (that is a View on the DataSet).

Instantiate and invoke a Web service or component

Instantiate and invoke a Web service

To use an XML Web Service a proxy class must be created by using the **Add Web Reference** command in the Project menu of the Visual Studio.NET or by using the WSDL.EXE command from the command line.

Using the Add Web Reference displays a form where it is possible to find Web Services on the UDDI Registry, or by using the WSDL description of the service directly.



The generated proxy can be instantiated like any other class.

```
mssoapsampleserver.Calc sampleCalculator = new
                                mssoapsampleserver.Calc();
double result = sampleCalculator.Add(10,20);
```

The namespace, the name of the class and the name of the methods are taken from the WSDL description of the XML Web Service.

There are a set of properties and methods that are always generated. This allows you to control the Web Service client. For example, the Url property allows you to change the URL of the Web Service.

Instantiate and invoke a COM or COM+ component

From .NET managed code it is possible to call COM and COM+ components using the Runtime Callable Wrapper (RCW).

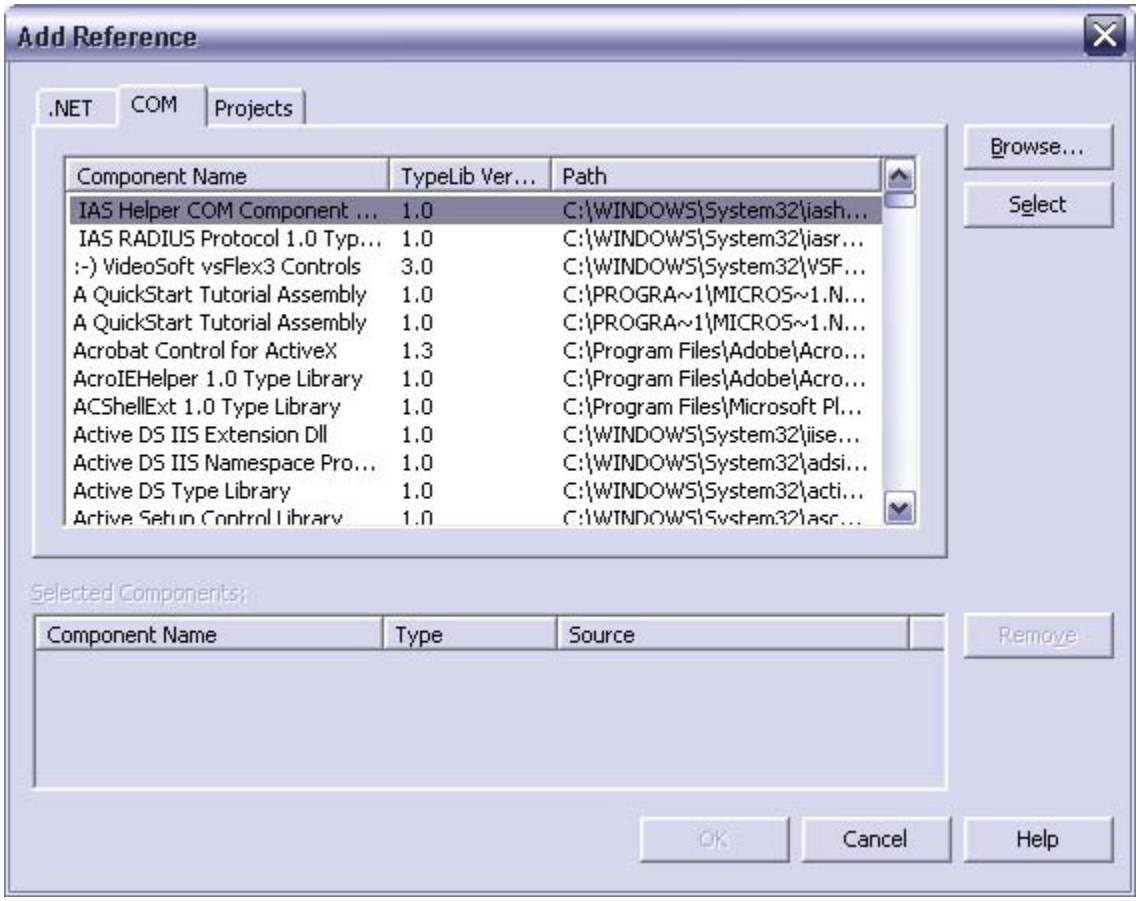
The RCW is created when a .NET Client loads a COM component. There is one RCW for each COM component, ensuring that they represent a single object identity. When the RCW is created, it reads the COM type library of the component and exposes every interface and every method of the component.

The RCW is responsible for keeping the COM object alive as long as the RCW is not garbage collected.

When a method is called on the RCW, it converts the parameters between .NET and COM types and vice-versa.

Then the RCW converts the HRESULT errors (if there are any) from the COM component in .NET exceptions, and converts the results into .NET types.

To generate the RCW, use the TLBIMP.EXE command-line utility or the Visual Studio.NET **Add Reference** command in the Project menu.



Instantiate and invoke a .NET component

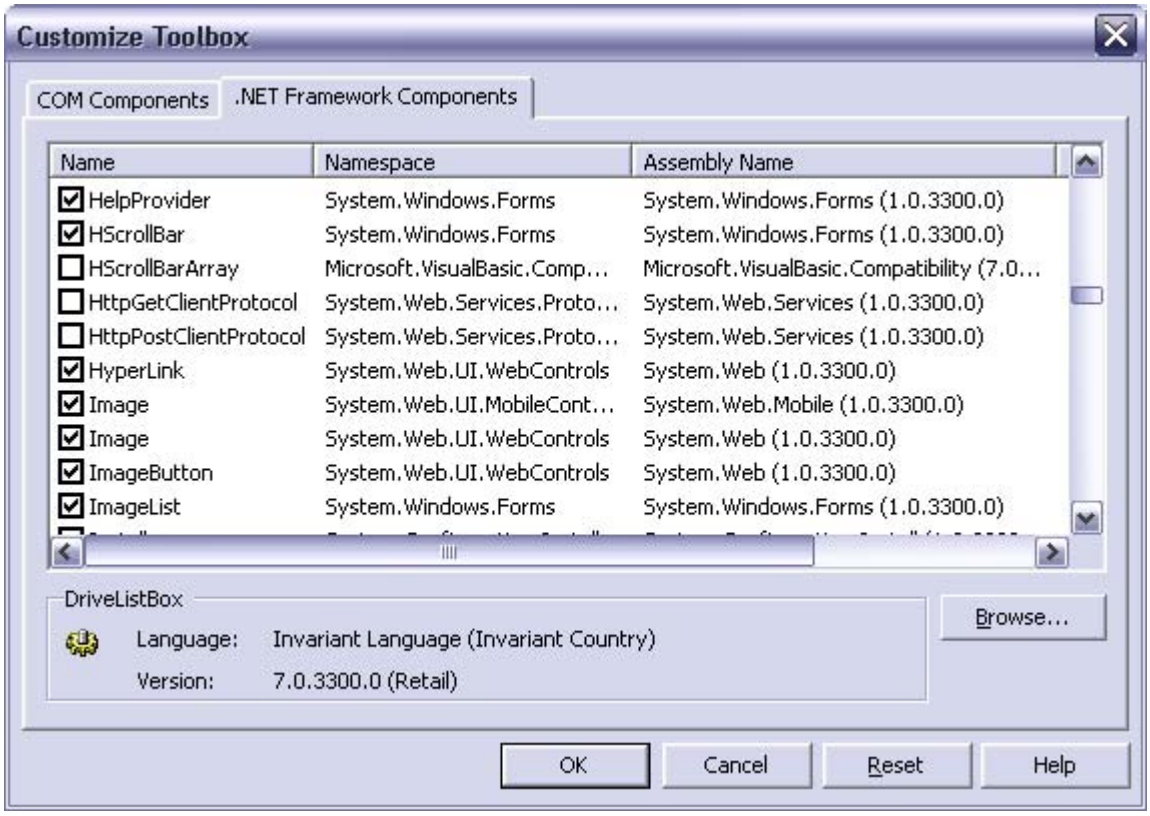
A .NET component is a class that implements the **System.ComponentModel.IComponent** interface (directly or by deriving it from another class that implements that interface).

The IComponent interface extends the **System.IDisposable** interface, which has a method called **Dispose** that is called to release external resources explicitly. When a component is created (via the **new** operator), the component is created in the garbage collected heap.

When there are no more references to the component, the garbage collector marks the component and, when the memory is needed, the finalizer is called.

If the component holds references to some expensive resources, these resources are taken until the component is finalized. The Dispose method can be explicitly called to release those resources and free them.

A .NET component can be included by using the Add Reference menu of the Visual Studio.NET or by customizing the Toolbox. Every .NET component has the RAD support built in the Framework, without requiring other code from the developer.



Call native functions by using platform invoke

To call functions in unmanaged code use the **DllImport** attribute.

```
[DllImport("KERNEL32.DLL", EntryPoint="MoveFileW",
SetLastError=true,
CharSet=CharSet.Unicode, ExactSpelling=true,
CallingConvention=CallingConvention.StdCall)]
public static extern bool MoveFile(String src, String dst);
```

DllImport must be placed on every method declaration, and has five parameters:

- "Name and location" of the DLL that contains the function that is called
- EntryPoint – Is the name of the entry point in the DLL
- CharSet – Is the character set used in the entry point. The default value is CharSet.Auto
- ExactSpelling – Indicates if the EntryPoint must exactly match the spelling of the entry point. The default value is False
- CallingConvention – Indicates the calling convention. The default value is CallingConvention.Winapi

DllImport is part of the namespace System.Runtime.InteropServices and it must be declared with the **using** keyword, or must be added to the name of the attribute.

Implement globalization

The .NET Framework supports multiple cultures and languages. The **CultureInfo** class contains culture specific information like:

- Language
- Country/region
- Calendar
- Etc…

Each culture has a unique name formed by two lowercase letters called culture code (the language; i.e., en, it, fr, de, etc.) and, optionally, two uppercase letters that specify the country or the region (i.e., IT, CH, etc.)

Implement localizability for the UI

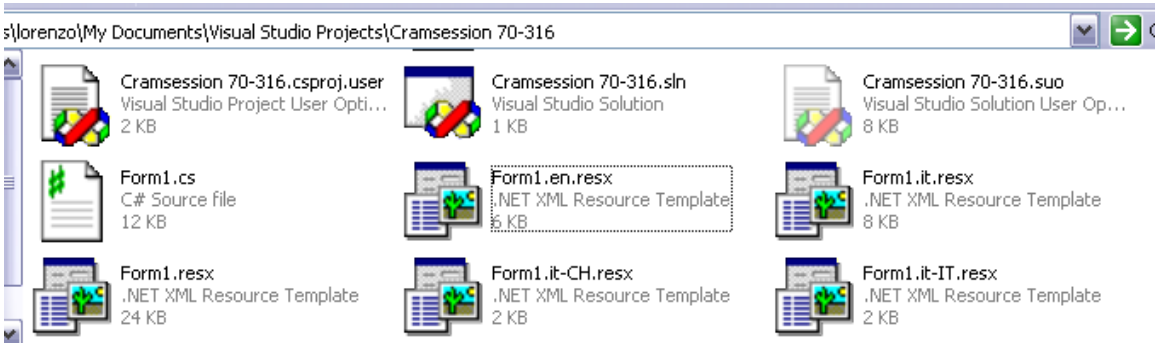
Every Form has two attributes related to localization:

- Localizable – specifies if the Form supports localization
- Language – specifies the language of the current Form

If Localizable is True, when the user specifies another Language, Visual Studio.NET will create a new resource file for that language.

Modifications made in a language are not seen by other languages.

Looking with Windows Explorer in the project directory shows that there are several resource files, the default one and one for each selected language, in this case English, Italian, Italian (Italy) and Italian (Switzerland).



When the application runs, it switches to the default locale.

To set the current locale, this statement must be inserted in the Main:

```
Thread.CurrentThread.CurrentUICulture =
Thread.CurrentThread.CurrentCulture;
```

With this statement, you read the settings stored in the Control Panel. (`Thread.CurrentThread.CurrentCulture`) and set it in the current User Interface.

Prepare culture-specific formatting

MSDN – Formatting for Different Cultures

Generally the ToString method is used to convert any type into a string.

There are overloaded versions of the ToString method that allow specifying the format to use to convert the type into the string. The first method allows you to specify a format string (like "C" for currency), and uses the current culture. The other methods allow specifying an IFormatProvider derived class (like CultureInfo, DateTimeFormatInfo and NumberFormatInfo) that can specify another culture's information.

```
public string ToString(string);

public string ToString(IFormatProvider);

public string ToString(string, IFormatProvider);
```

For example, it's possible to format a number using a different culture other than the current one.

```
int MyInt = 100;
CultureInfo ItalianCulture =  new CultureInfo("it-IT");
String MyString = MyInt.ToString("C", ItalianCulture);
MessageBox(MyString);
Console.WriteLine(MyString);
```

If the string is printed on the Form, the € character is printed, but it is not printed on the Console because it's not supported in the console window.

Create, implement, and handle events

MSDN – Event Handling in Windows Forms

In a typical graphics environment like Windows, applications are "message driven".

A program starts, prepares the main Form, initializes it and then waits. All the functions of the program are executed in response to messages.

Those messages are generated by the operating system when an event occurs, like mouse clicks, keystrokes, window movements, timers, etc…

Those messages are handled by the .NET Framework that generates an event for every message received.

Events are handled by event handlers.

Suppose that you want to handle the Paint event on a Form. You have to look at the PaintEventHandler delegate that defines the signature of the method that will be called by the system:

```
public delegate void PaintEventHandler(object objSender, PaintEventArgs
pea);
```

this delegate is defined in the System.Windows.Forms namespace.

Then you have to implement an event handler with the same signature as the delegate:

```
static void MyPaintEventHandler(object objSender, PaintEventArgs pea)
{ … }
```

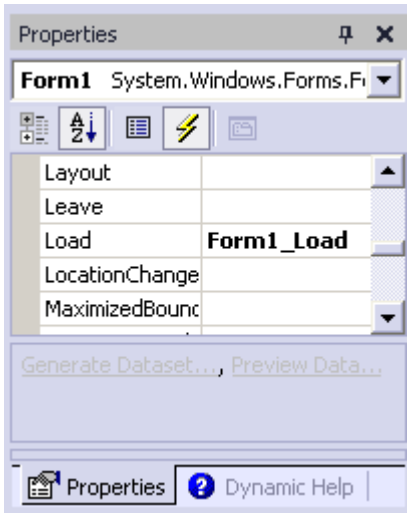
The last thing is to bind the event handler to the event:

```
form.Paint += new PaintEventHandler(MyPaintEventHandler);
```

An event can be bound and unbound with the += and -= operators.

To define a custom event the same operations are required: defining the delegate, creating the event handler, and binding the event with the event handler.

Events handler can be also set in the Events tag () of the Property Window:



Implement print capability

There are five steps to add print capability to an application:

1. Add the PrintDocument object to the application (the PrintDocument object is in the Toolbox window).
2. Write the PrintPage event handler to construct the content of the print document.
3. In the PrintPage event handler use the PrintPageEventArgs parameter to store and retrieve information about the print document.
4. Use the standard Print Dialogs (PrintPreviewDialog, PageSetupDialog,PrintDialog) to interact with the user.
5. Print the document using the PrintDocument.Print method.

Implement accessibility features

Accessibility features are implemented trough specialized programs like the Narrator, the Magnifier, the On-Screen Keyboard and other 3rd party products.

It's important to specify standard properties like Text, Font Size, Forecolor, Backcolor and BackgroundImage with correct values. For example, use the ampersand "&" character in the Text property to create access keys.

There are other properties like:

- **AccessibleName** – is the text that describes the content of the control or of the form. It's read by the Narrator
- **AccessibleDescription** – is the text that provides greater context information on a control. It's not read by the Narrator, but can be read by other programs
- **AccessibleRole** – specifies the role of the control in the user interface (like button, text, label, etc.) The default value is enough in most situations

Creating and Managing Components and .NET Assemblies

Create and modify a .NET assembly

Create and implement satellite assemblies

MSDN- Creating Satellite Assemblies

Satellite assemblies contain only resource files and no code. Satellite assemblies can be replaced or updated without modifying the main assembly.

Satellite assemblies can be generated with Visual Studio.NET or via the Assembly Linker (al.exe).

Create resource-only assemblies

Go to Project Menu and select Add New Item, and then choose Assembly Resource File.

If you want to create a localized resource file, add the culture identifier (i.e., it-IT or de-CH) to the resource name.

To use a resource simply create a ResourceManager:

```
ResourceManager resMan = new
ResourceManager("Namespace.Resource",
Assembly.GetExecutingAssembly());
```

and then get all the information from the resource:

```
MessageBox.Show(resMan.GetString("strMessage"));
```

If there are localized versions of the same assembly, the right one is selected.

If there isn't the correct resource file, the default one is loaded.

Create a Windows control

There are several ways to create a Windows control:

- Extending an existing control

  Extending an existing control allows you to add functionalities or to customize an existing control. It's the right choice when the new control is very similar to an existing control.

  To extend an existing control, the new control must inherit from it. Then it can override all the properties, methods and events.

  Remember to implement the constructor in the inherited object because it's not inherited. If there are no changes, the new constructor can call the constructor of the base object:

  ```
  Public NewTextBox():base();
  ```

- Combining existing controls

  Composite controls are built by combining different existing controls into one. All composite controls derive from System.Windows.Forms.UserControl.

  The UserControl class extends the ContainerControl class, which allow the containment of other controls (also the Form class derives from the ContainerControl class).

- Creating a custom control

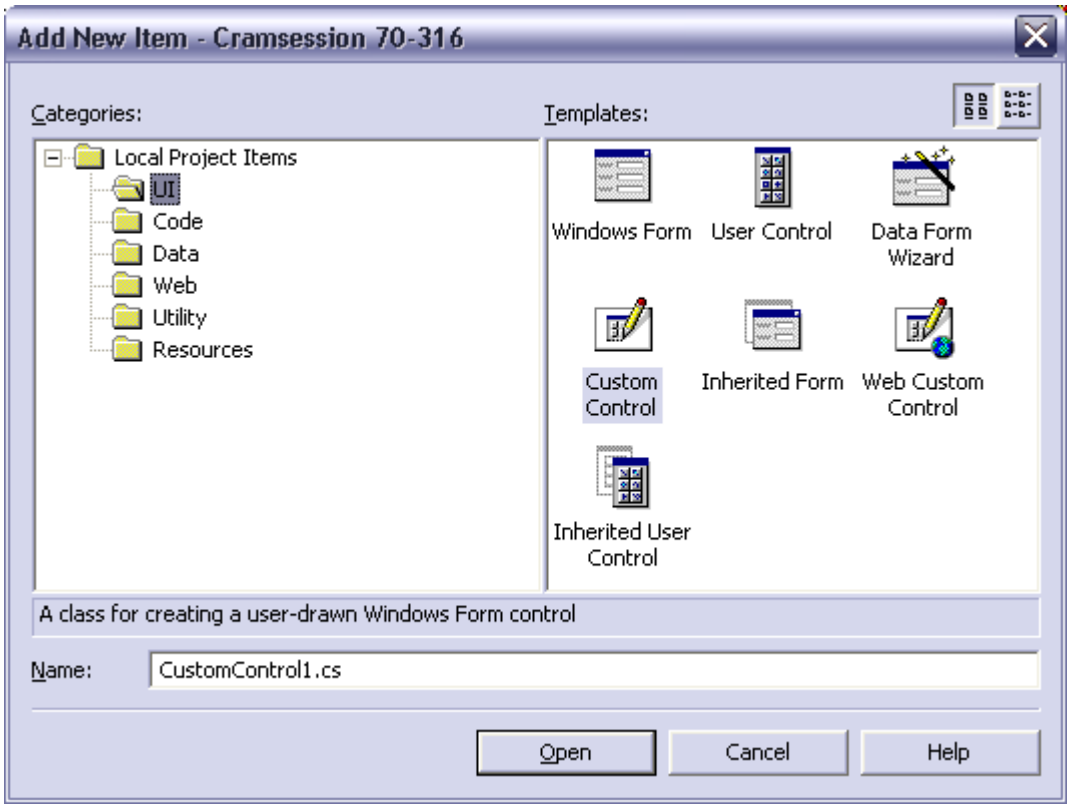
  A custom control doesn't derive from or combine existing controls.

  A custom control handles the OnPaint event to draw itself and must handle all the other events, and must implement all the methods, properties, and so on.

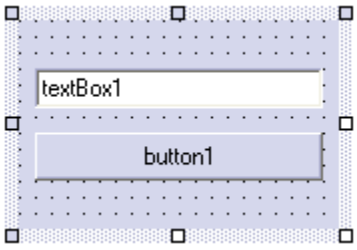
  Typically a custom control derives from System.Windows.Forms.Control, which provides all the plumbing needed to manage message routing, handles mouse, keyboard and other events, and adds a set of common properties like ForeColor, BackColor, Height, Width, etc…

Create a Windows control by using visual inheritance

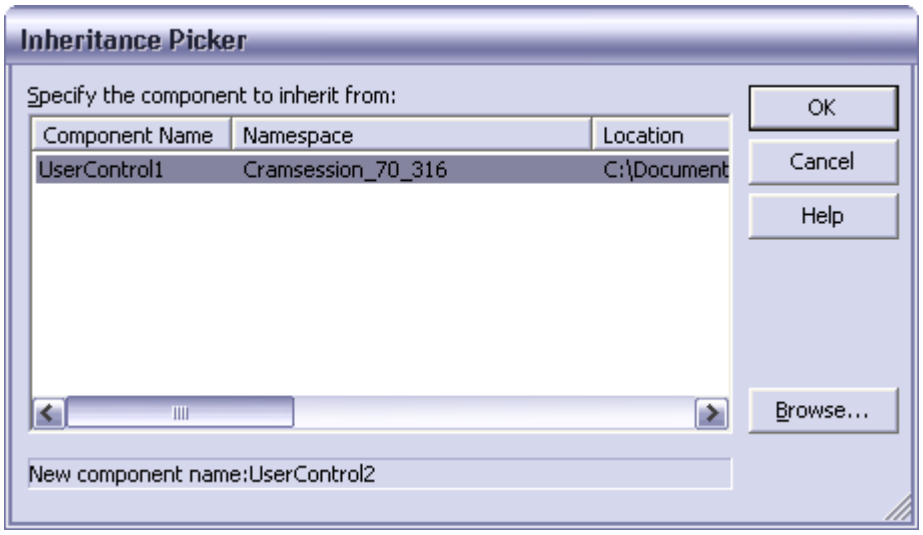
It's very easy to create a new control in Visual Studio.NET. Simply select Add New Item from the File Menu and go to the UI subdir.



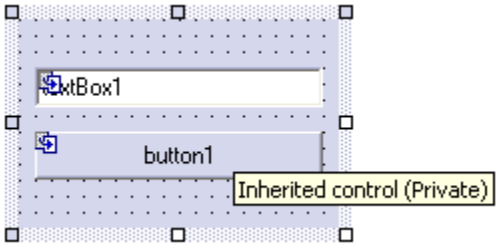
User Control allows creating a composite control. A blank container is displayed, and existing controls can be added to it. The container can be resized.



Inherited User Control allows you to extend an existing control. The Inheritance Picker is displayed, and the programmer can select from which existing control the new control must inherit.



Then a new file is created, and the new control can be displayed in the designer:



The new control can modify properties, event handlers, methods, and appearance of the old control.

Custom Control is self-explanatory. The most important thing is that the Wizard creates the class (that inherits from Control), the empty constructor, and the OnPaint event handler that calls the OnPaint of the base class.

Host a Windows control inside Microsoft Internet Explorer

The .NET Framework supports hosting applications and controls in Internet Explorer 5.5 or later. An application hosted by Internet Explorer is essentially an application that runs within a Web page. The .NET Framework Redistributables must be installed on the client machine.

To load a .NET control, the **OBJECT** tag in a HTML page is used.

OBJECT is the same tag used to host an ActiveX control inside an HTML page, but some attributes are different. The OBJECT tag, which is used to call a .NET control, uses the CLASSID attribute to specify the component name, the namespace and the name of the control with the syntax:

```
component-name#namespace.control-name
```

An example is:

```
<OBJECT id="control-id" classid="MyCcontrol.dll#MyNamespace.MyControl"
width="..." height="...">
  <PARAM name="MyParam" value="...">
</OBJECT>
```

The OBJECT tag cannot be used to host a standard Windows Forms control in IE, because it can be used only to host custom or user controls.

The control is loaded form the Web Server (the current directory or its "bin" subdir).

Consuming and Manipulating Data

Access and manipulate data from a Microsoft SQL Server™ database by creating and using ad hoc queries and stored procedures

To access and manipulate data from a Microsoft SQL Server database, the Transact SQL language is used.

The most important commands are:

- SELECT – used to retrieve data stored into the database
- INSERT – used to insert new rows into the database
- UPDATE – used to change data in the database

Stored procedures are T-SQL commands stored into the database for later use. Using stored procedures to access a SQL Server database gives these advantages:

- SQL statements are pre-compiled and execute faster
- Syntax is checked at compile time, not every time
- Stored Procedures remain in the cache of the database
- Network traffic is low because the code is in the database
- Stored Procedures can be shared between applications, allowing code reuse
- Stored Procedures provide better security

Access and manipulate data from a data store. Data stores include relational databases, XML documents, and flat files. Methods include XML techniques and ADO .NET

ADO.NET is the preferred technique to access data in the .NET Framework.

ADO.NET supports various types of data storage:

- Relational databases (SQL Server, Oracle, etc…)
- Hierarchical data (XML files)
- Structured data (CSV, Excel, Active Directory, etc…)
- Unstructured data

ADO.NET supports two different environments:

- Connected
- Disconnected

A connected environment requires a continuous connection with the data source. A disconnected environment allows retrieving data, releasing the connection, manipulating the data, reopening the connection and storing the data back into the data source.

ADO.NET is composed of a series of classes, interfaces, structures and enumerations grouped in a series of namespaces:

- System.Xml
- System.Data
- System.Data.Common
- System.Data.SqlTypes
- System.Data.SqlClient
- System.Data.OleDb

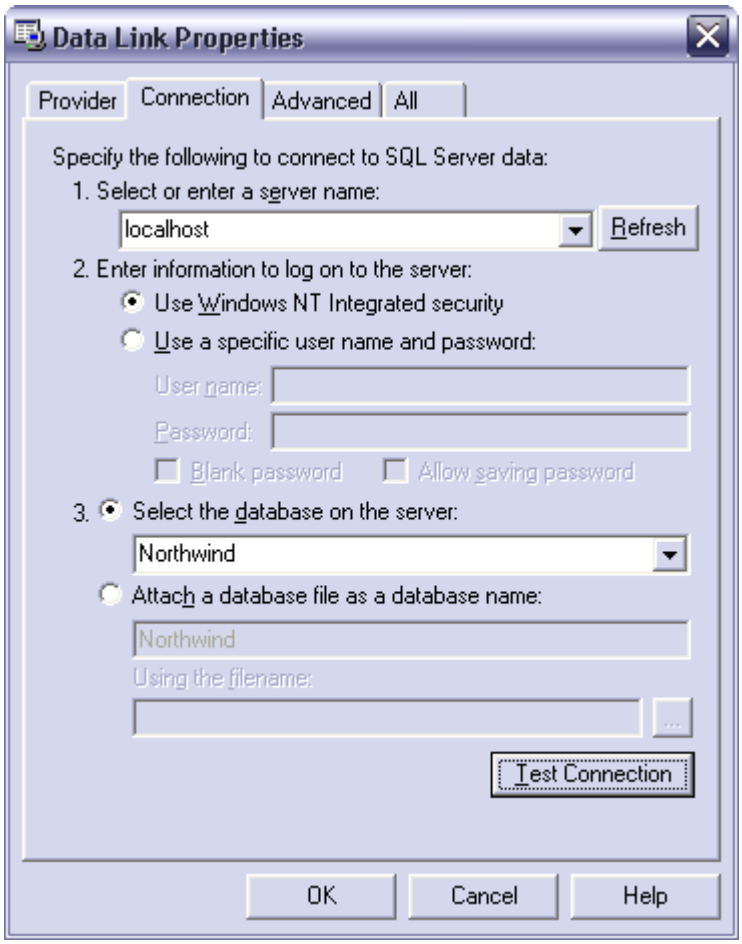
The last two namespaces are used to access data. The first is to access SQL Server 7.0 and the second is for all the other OLE DB data sources (ODBC, Oracle, DB2, MySql, etc…).

To access data in a connected environment, three classes are involved:

- xxxConnection – used to establish a connection to a specific data source
- xxxCommand – used to execute a command from a data source
- xxxDataReader – used to read a forward-only, read-only stream of data

where **xxx** can be **Sql** for the SqlClient data source and **OleDb** for the OleDb data source.

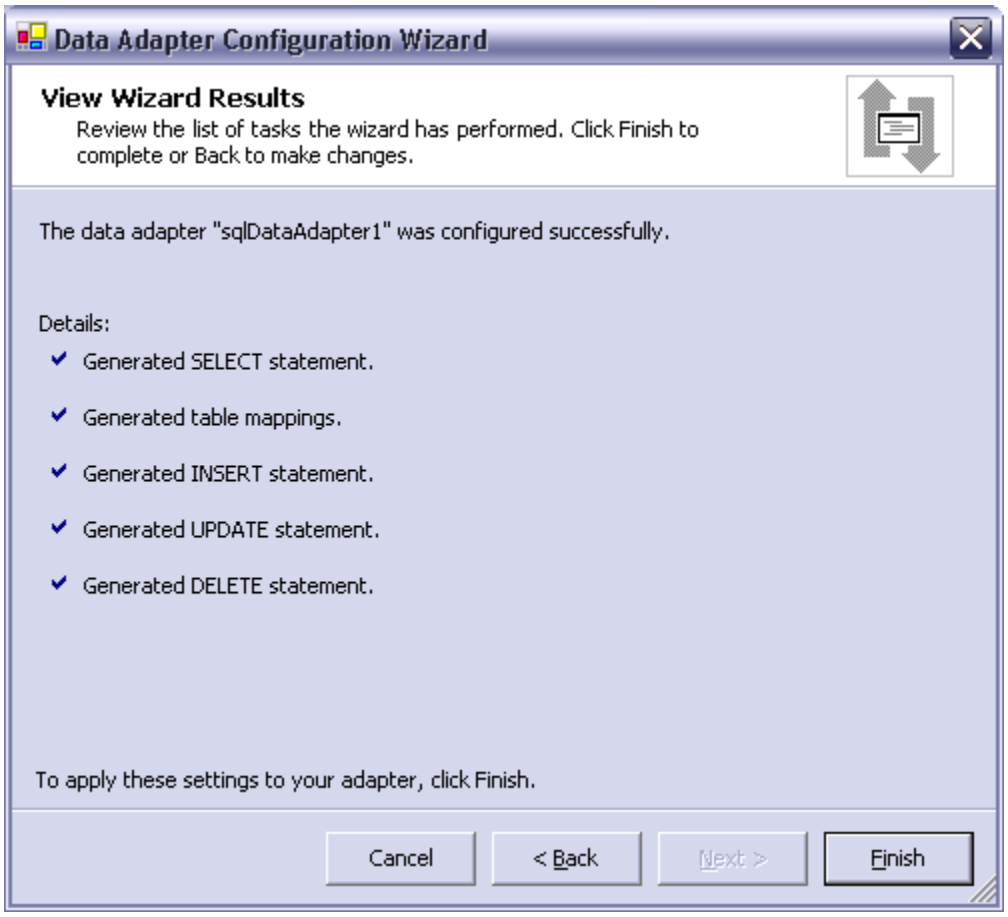
A wizard helps to set up a connection to the database:



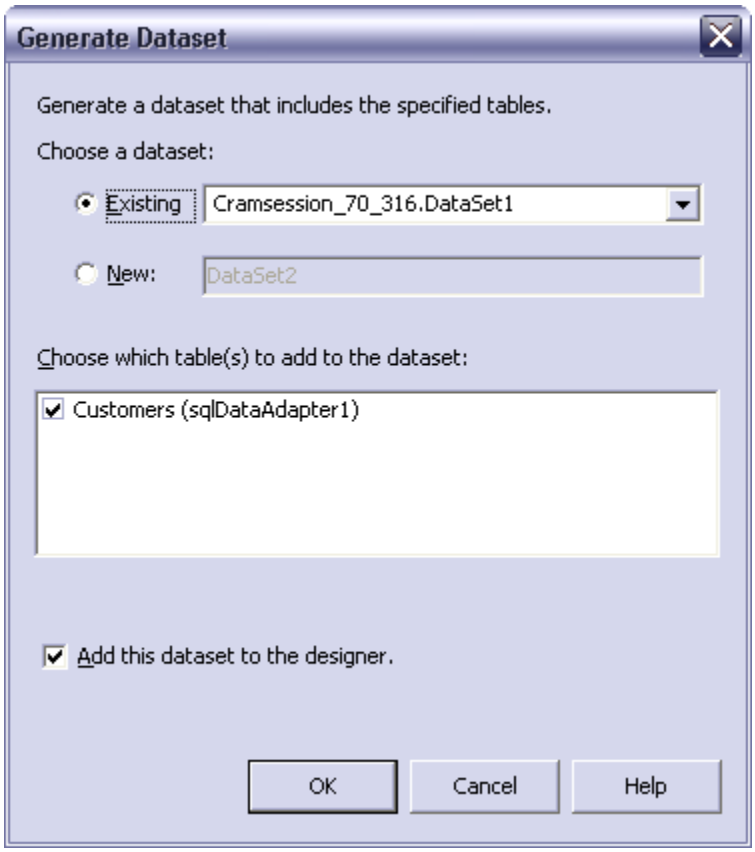
To access data in a disconnected environment, two other classes are involved:

- xxxDataAdapter – uses the xxxConnection, xxxCommand and xxxDataReader classes to populate a DataSet and to update the data source with the changes made to the DataSet
- DataSet – is a class that encapsulates a set of tables and their relationship. Every operation made to the DataSet can be replicated to the data source using the xxxDataAdapter.

There is a wizard that helps to configure a DataAdapter:



After the data adapter is created, it can be used to obtain a DataSet with a wizard:



XML is an important part of data access, because it's possible to retrieve an ADO.NET DataSet and see it as an XML document or vice versa. It's also possible to fill a DataSet with XML data.

Handle data errors

ADO.NET classes throw exceptions if something goes wrong.

Every .NET data provider has an exception class:

- SQLException (has an Errors collection that contains SqlError objects)
- OleDbException (has an Errors collection that contains OleDbError objects)

It's important to enclose ADO.NET code in **Try..Catch..Finally** statements to properly handle errors, and to execute the clean-up code when needed.

The **xxxError** object has some properties like:

- **Class** – the severity level of the error
- **LineNumber** – the line number within the SQL stored procedure or batch file
- **Message** – the text that describe the error
- **Number** – the error code

Testing and Debugging

Create a unit test plan

With unit testing, you test every unit separately from the rest of the application to determine if it behaves as expected.

The most common approach to unit testing requires the implementation of drivers and stubs. Drivers are used to simulate a call from another unit, whereas stubs are used to simulate a called unit.

Implementing drivers and stubs is expensive, and sometimes unit testing is not implemented or has a low priority in the project.

This is a great mistake, because it's been proven that a large percentage of bugs are identified during these kinds of tests.

Trying to discover errors across the entire application is more complicated than finding them in every single unit.

Implement tracing

There are two classes used to trace debugging or other information when the program runs:

- Debug – is used to print debugging information when the project is build in Debug mode
- Trace – is used to print information also when the program is in Release mode. Traces can be enabled or disabled with trace switches

Both classes share a set of methods:

- **Write**, **WriteLine** – they always send the information to the trace listener
- **WriteIf**, **WriteLineIf** – they send the information to the trace listener if the condition is true
- **Assert** – tests a condition and stops the program execution if it's false
- **Fail** – causes a termination of the program

Add trace listeners and trace switches to an application

Trace listeners are used to collect, store and route tracing messages.
Tracing messages can be logged, printed or stored on a file.

Listeners can be used with both Debug and Trace classes. Three types of listeners are already implemented in the .NET Framework:

- **DefaultTraceListener** – it's the default behavior of both Trace and Debug classes. It sends messages to OutputDebugString and calls the Debugger.Log method (in Visual Studio.NET this sends the messages to the Output window). In addition to this behavior, Fail and Failed Assert generate a message box.
- **EventLogTraceListener** – it sends the messages to an event log.
- **TextWriterTraceListener** – it sends the messages to an instance of the TextWriter class or to a Stream derived class.

To add a listener to Debug or Trace classes simply create it and add it to the Listener's collection of one of them.

Here is an example:

```
System.IO.FileStream myTraceLog = new
   System.IO.FileStream("C:\\myTraceLog.txt",
   System.IO.FileMode.OpenOrCreate);

TextWriterTraceListener myListener = new
TextWriterTraceListener(myTraceLog);

Trace.Listeners.Add(myListener);
```

It's also possible to create a Listener and use it without adding it to the Debug or Trace classes.

Simply use the Write or WriteLine methods of the listener, but remember to call the Flush method at the end. This is not required if the listener is added to the Listeners collection.

Trace switches are used to enable or disable tracing. There are two types of predefined switches:

- **BooleanSwitch** – enables or disables tracing
- TraceSwitch – enables or disables tracing based on the level of the message (Off, Error, Warning, Info, Verbose, in increasing order). When a level is selected, all the lower levels are selected

Here is an example:

```
TraceSwitch myTraceSwitch = new TraceSwitch("MySwitch",
"Description");
myTraceSwitch.Level = TraceLevel.Info;
Trace.WriteLineIf(myTraceSwitch.TraceError, "It's displayed
because it's lower then Info");
```

Trace switches can also be configured with the XML configuration file using:

```
<system.diagnostics>

   <switches>
```

```
        <add name="MySwitch" value="0" />
      </switches>
    </system.diagnostics>
```

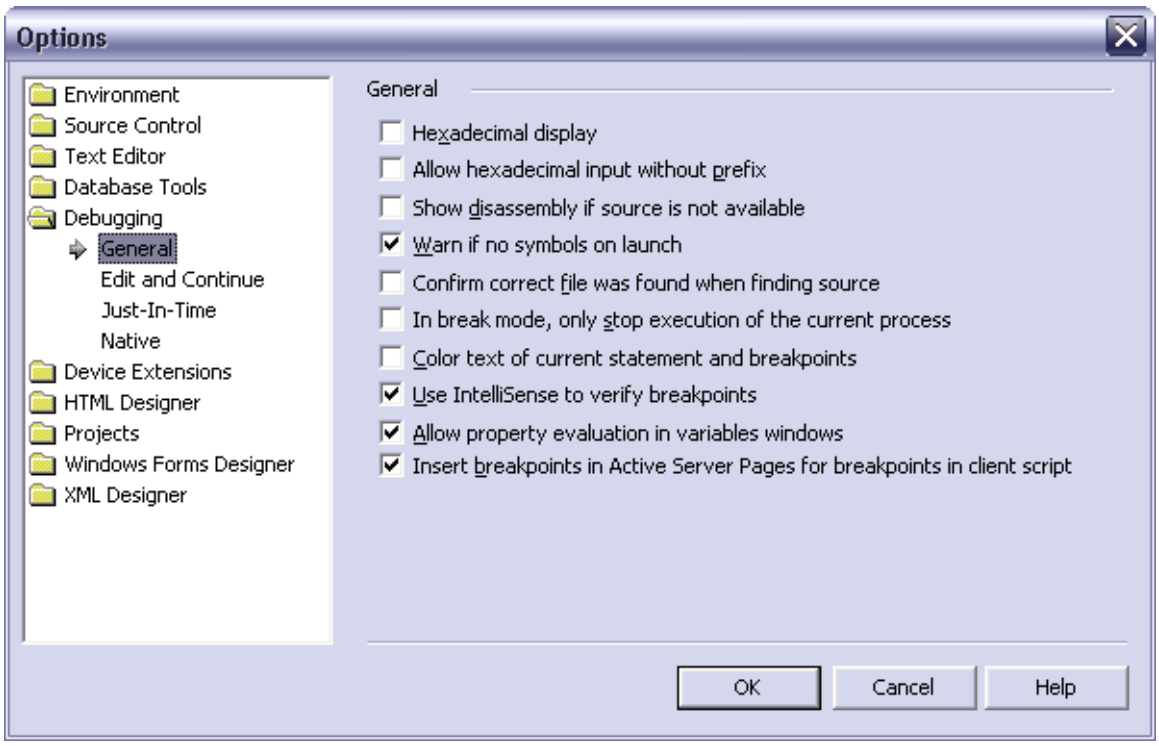
Where 0 corresponds to Off and 4 to Verbose.

Debug, rework, and resolve defects in code

Configure the debugging environment

To set the debugging environment use Tool menu, Options and then go to Debugging folder.

Most of the options are under General:



In the Edit and Continue tab it's possible to enable the feature for each .NET language.

Just-In-Time enables the Just-In-Time debugging.

Native enables RPC debugging and Load DLL export.

Create and apply debugging code to components and applications

Debug methods are the same as the Trace class.

Provide multicultural test data to components and applications

It's very difficult to manually enter test inputs in different languages (Arabic, Hebrew, Thai, East Asian languages, German, Greek, etc…). In some cases there are limitations that are not trivial to understand.

It's better to use an Unicode text generation utility that can generate text according to different languages' rules.

It's also very important to test how the application's UI is rendered in different cultures. English strings are generally shorter than those of any other language.

Execute tests

It's important to have some test machines that allow testing for different configurations. For example, if the application is localized it's important to test it with different user locale, like German to test Western Europe characters, and Japanese to test East Asian characters.

Also, a Multi User Interface version of Windows 2000 could help, because it can change the appearance of the UI without having to reinstall the OS.

Resolve errors and rework code

With Edit and Continue enabled it is possible to modify code while debugging it. Thus it can be tested directly with the data that have caused the problem.

Deploying a Windows-based Application

Plan the deployment of a Windows-based application

There are several options for packaging and deploying a .NET application.

Packaging:

- **Executables and DLLs in their original folder hierarchy** – no packaging
- **Windows Installer 2.0 files** – executables and DLLs are packaged into a single .msi file
- **Cabinet files** – executables and DLLs are packaged into a single .cab file, that can be downloaded into an HTML page

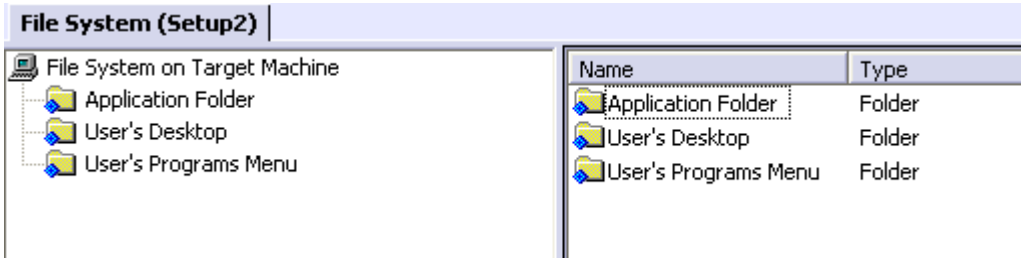
Deploying:

- **XCOPY or FTP setup** – simply copy the folder hierarchy to the destination folder and execute the program. Doesn't allow you to access the GAC (Global Assembly Cache - see later)
- **Windows Installer setup** – the program comes with an installer that allows to install, repair and remove all the files and also access to the GAC
- **Code download** – the program can be launched from an Intranet or from the Internet (depending on the security permissions), in the form of a CAB file or directly from the executable

Create a setup program that installs an application and allows for the application to be uninstalled

Use the Setup Project (see later) to create a new Windows Installer file.

It's possible to add files to the Application Folder, to the user's desktop and to the user's Programs menu. It's also possible to add other special directories to the project.



After building the solution, some of these files are generated:

where InstMsiA.exe, InstMsiW.exe, Setup.exe and Setup.Ini are the Windows Installer 2.0 installation files and Setup2.msi is the created file.

Register components and assemblies

To register .NET components and assemblies it's possible to use the component selector (right-clicking Add->Assembly on one of the file system folder) and add them to the installation package.

Perform an install-time compilation of a Windows-based application

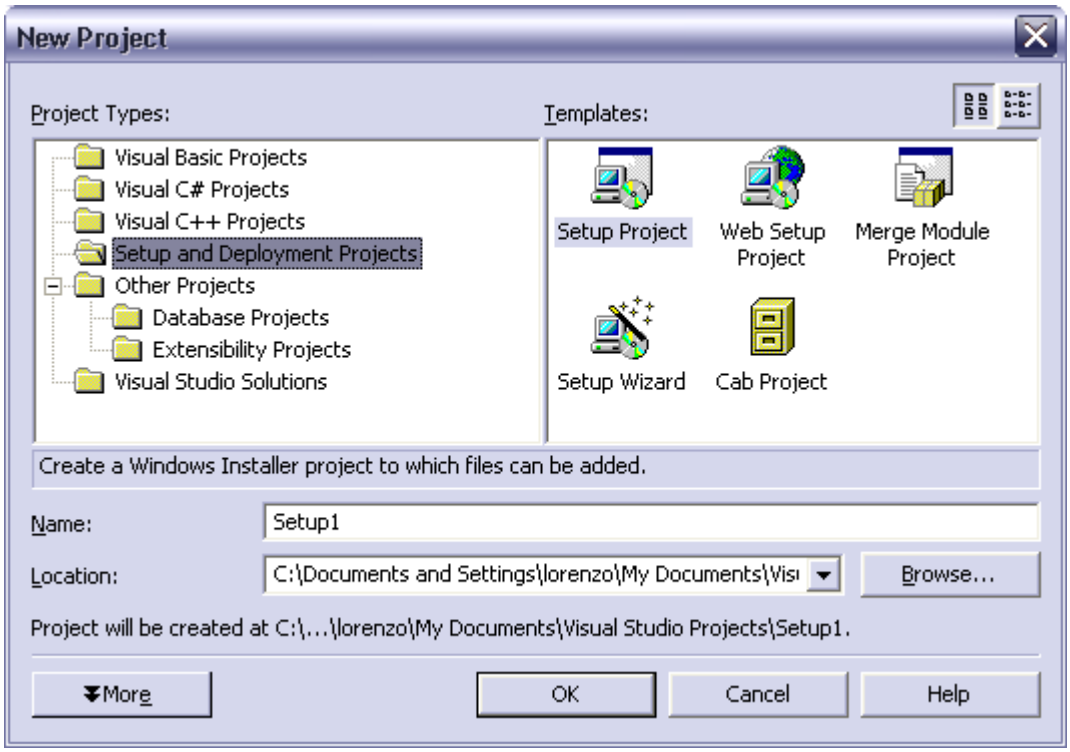
NGEN.EXE is the utility that precompiles an application so it can start faster.

Precompiling doesn't allow the run-time optimizations that the CLR uses, but for big programs it allows a reduced start-up time.

Remember that precompiling doesn't allow distributing an application without the .NET Redistributables that are always required on the user's machine.

Deploy a Windows-based application

Use setup and deployment projects



There are several projects available:

- Setup Wizard – allows using a Wizard to create one of the following projects
- Setup Project – allows creating a Windows Installer package. It's an .msi file that can be run from a removable media or a network share
- Web Setup Project – allows creating a setup package that can run directly from a Web Server
- Merge Module Project – allows creating a module that modifies an existing Windows Installer project. It's used to deploy patches, new versions, new languages or add-ons

- Cab Project – allows creating a CAB file that can be downloaded from an HTML page

Add assemblies to the Global Assembly Cache

The Global Assembly Cache (GAC) is used to store assemblies that can be shared among many applications. It can store one or more versions of the same assembly (side-by-side deployment).

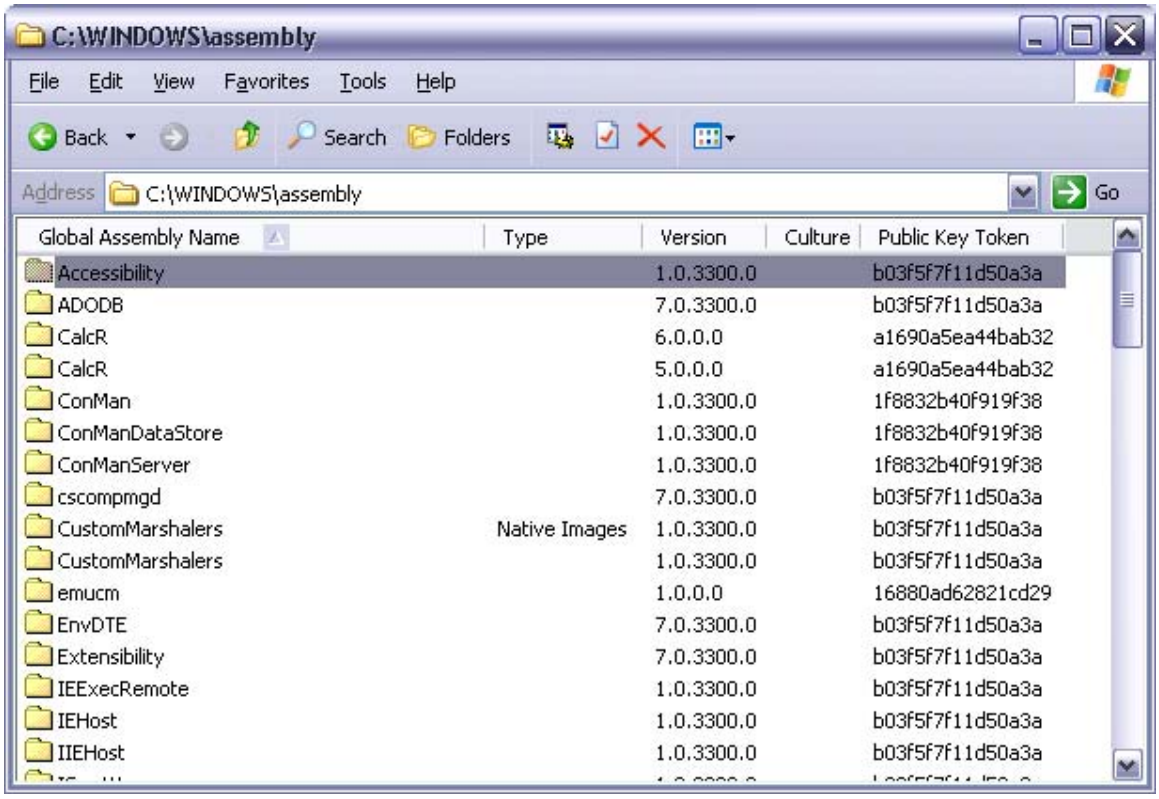
Assemblies must have a Strong Name because of the strict naming rules used to ensure that the correct assembly is loaded when requested.

Administrator privileges are required to install or uninstall an assembly in the GAC. Windows Installer, or the .NET Framework configuration tool (MSCORCFG.MSC), can be used to install assemblies.

In development environments, the GACUtil can be used for testing purposes. GACUtils options are:

- /i or –i install an assembly into the GAC
- /l or –l list the contents of the GAC
- /u or –u uninstall one or more assemblies

The .NET installer loads a shell-extension that displays extended information on the assemblies on the GAC in Windows Explorer:



Verify security policies for a deployed application

Security policies are a configurable set of rules that the CLR uses when it loads the code to determine if it can run. The CLR ensures that the code can access only the resources allowed by the policies.

Security policies can be configured by editing the XML configuration file or by using the .NET Framework Configuration Tool:

Launch a remote application (URL remoting)

An application can be launched from the local harddisk, from an Intranet share (\\server\share\remoreApp.exe), or from the Internet, depending on the security policies. There are several zones with different Security Levels:

- **Full Trust** – no security checks are performed
- **Medium Trust** – programs cannot access the registry or other protected resources and cannot access the local file system without the permission of the user but can call back the original site, resolve domain names or access windows resources
- **Low Trust** – programs cannot access local resources and have limited access to windows resources or file system
- **No Trust** – programs are not executed in this zone

Security zones can be configured with a Wizard:

Maintaining and Supporting a Windows-based Application

Optimize the performance of a Windows-based application

There are several ways to improve performance of a .NET application. Look at

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetperftips.asp?frame=true#dotnetperftips_topic2

and http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnetperftechs.asp?frame=true

One of the most important techniques is precompiling. For applications that have a lot of forms, precompiling can boost start-up times.

Diagnose and resolve errors and issues

Use Tracing to log errors and issues, so the user can submit error information. This makes it easier for you to understand the situation that has lead to the error.

Remember that Tracing can be enabled or disabled in the application's configuration file.

Configuring and Securing a Windows-based Application

Configure a Windows-based application

The .NET framework allows to configure Windows-based applications by using a configuration file in the application's directory, with the application's name plus the .config extension. For example, the myApplication.exe could have a configuration file named myAppliation.exe.config.

Configuration files are XML files (case-sensitive) that allow you to configure application's settings; for example, tracing, or if the application uses special version of an assembly, etc.

Most important tags used to configure an application are:

- <configuration> is the root element.
- <runtime> is a sub element of <configuration> and contains information about assembly binding at run-time.
- <assemblyBinding> is a sub element of <runtime> and contains information about assembly version redirection and location of assemblies.
- <dependentAssembly> is a sub element of <assemblyBinding> and is used to encapsulate information about each assembly.
- <assemblyIdentity> is a sub element of <dependentAssembly> and contains information used to identify an assembly.

- <bindingRedirect> is a sub element of <dependentAssembly> and is used to redirect the CLR from one assembly version to another.

- <codeBase> is a sub element of <dependentAssembly> and specifies the position of a strong-named assembly.

Configure security for a Windows-based application

There are two ways to handle security in .NET:

- Code Access Security – code can access protected resources only if it has permissions to do so

- Role-based Security – code can determine the identity and the role of the user and access resources according to this role

Code access security is based on three main concepts:

- **Evidence** – is a set of information about the identity and the origin of an assembly, like the assembly strong name, the publisher signature, the originating zone, the hash, etc…

- **Permissions** – are rights to access protected resources, like FileIOPermissions, IsolatedStorageFilePermissions, PrintingPermissions, SqlClientPermissions, UIPermissions, WebPermissions, etc…

- **Security Policies** – are used to map Evidence with Permissions. For example, an assembly that comes from a trusted site can do only a certain set of operations. There are several Security Policies: Enterprise, Machine, User, Application Domain, and the most restrictive rules of these policies are applied.

Role-based security is based on two main concepts:

- **Identity** – normally the user's log on name

- **Principal** – roles associated with a user, like Windows user and groups or custom roles

An authenticated identity means the Identity and the Principal of one user together.

Authentication is the process of finding and verifying the identity of one user (using NTLM, Kerberos or other methods).

Authorization determines if an authenticated user can or can't perform an operation.

Role-based security is based on classes and methods that allow checking the user's identity and determining if the user has the power to do an operation.

Special thanks to
Lorenzo Barbieri
for contributing this
Cramsession.