Introducere in Windows Forms

Multe dintre sistemele de operare creaza evenimente si utilizeaza ferestere pentru a interactiona cu utilizatorii. In ceea ce priveste Microsoft Windows, pentru a crea o fereastra sunt utilizate rutinele bibliotecii Win32.

Arhitectura .NET contine un numar impresionant de clase, structuri, interfete, metode etc. (Base Class Libraries) utile pentru crearea de aplicatii, iar unele dintre acestea sunt destinate pentru a crea ferestre. Ceea ce in programarea Windows se numeste o fereastra, in programarea .NET se numeste form. Acest form este caracterizat de un titlu, contine bare de navigare, un meniu, toolbars (toate acestea sunt optionale) iar ceea ce ramane se numeste zona client.

Pentru a crea un form creati in fapt un obiect de tipul unei clase care mosteneste clasa Form din spatiul de nume System.Windows.Forms. Clasa Form este derivata din ContainerControl care este derivata la randul ei din ScrollableControl. In fapt, lantul mostenirilor este:

Object-MarshalByRefObject-Component-Control-ScrollableControl-ContainerControl-Form

Form-urile sunt obiecte care expun proprietati ce definesc modul de afisare si metode care raspund evenimentelor ce definesc interactiunea cu utilizatorul. Setand proprietatile unui form si scriind codul ce raspunde la evenimente, in fapt realizam obiecte care se comporta conform cerintelor cerute de aplicatie. Form-urile sunt instante ale clasei Form (sau ale unor clase derivate din aceasta).

Cele mai simple exemple

In primele doua exemple se creaza obiecte instanta (forms) ale clasei MyForm si se utilizeaza metoda Show( ) si respectiv proprietatea Visible pentru a face vizibile ferestrele (form-urile) create. Atunci cand este rulat unul din primele doua exemple, programul arata form-ul si avanseaza la linia urmatoare, care este de fapt sfarsitul programului. Deoarece programul a ajuns la final, formul se inchide. Astfel, form-ul se observa doar pentru o fractiune de secunda.

Pentru a crea un form care sa ramana pe ecran se utilizeaza metoda Run() din clasa Application. Clasa Application face parte din acelasi spatiu de nume System.Windows.Forms, este derivata direct din clasa Object si toate metodele sale sunt statice.

Spre deosebire de metoda Show() care dupa ce arata form-ul si avanseaza la linia urmatoare, metoda Run() creaza o bucla (loop) care face ca programul sa ruleze in continuare pana cand este produs un eveniment care incheie bucla (de regula acest eveniment se produce prin apasarea unui buton Cancel).

Exemplul 1:

```
using System.Windows.Forms;
class MyForm:Form
{public static void Main()
{MyForm myfirstform=new MyForm( );
myfirstform.Show( );} }
```

Exemplul 2:

```
using System.Windows.Forms;
class MyForm:Form
{public static void Main()
{MyForm myfirstform=new MyForm( );
myfirstform.Visible=true;} }
```

Exemplul 3:

```
using System.Windows.Forms;
class MyForm:Form
{public static void Main(string [] args)
{MyForm myfirstform=new MyForm( );
myfirstform.Text="";
if (args.Length>0)
{ for (int i=0;i<args.Length; i++)
myfirstform.Text+=args[i]+" ";}
Application.Run(myfirstform);}
}
```

Crearea unui form

Clasa Form contine un numar mare de metode, proprietati, evenimente pentru a putea fi descrise intr-o carte. Functionalitatea acestora poate fi consultata prin documentatia online. Indicii privind functionalitatea membrilor clasei Form pot fi gasite si in mediul de programare Visual Studio .NET.

Este util sa studiem cateva dintre acestea. Vom incepe prin a descrie cateva proprietati asociate barei de titlu.

Bara de titlu (control box) ofera utilizatorului informatii despre program. In exemplele precedente s-a observat ca aceasta contine cativa itemi precum Minimize, Maximize, si Close. Acestia pot fi controlati prin intermediul proprietatilor:

ControlBox (determina daca control box este afisat), HelpButton (indica daca butonul de help este afisat. Acest buton este disponibil doar daca MinimizeBox si MaximizeBox sunt ambii false) MaximizeBox (indica daca butonul Maximize este inclus) MinimizeBox (indica daca butonul Minimize este inclus), Text (include titlul ferestrei).

Exemplul 4

```
using System.Windows.Forms;
public class FormApp : Form
{
   public static void Main( string[] args )
   {
   FormApp frmHello = new FormApp();
   // Caption bar properties
   frmHello.MinimizeBox = true;
   frmHello.MaximizeBox = false;
   frmHello.HelpButton = true;
   frmHello.ControlBox = true;
   frmHello.Text = @"My Form's Caption";

   Application.Run(frmHello);
   }
}
```

Dimensionarea unui form

Urmatorul lucru care trebuie controlat reprezinta dimensiunea form-ului. Se pot utiliza o serie larga de metode si proprietati care realizeaza acest lucru. Exemple de proprietati care gestioneaza dimensiunile unui form:

AutoSize The form automatically adjusts itself, based on the font or controls used on it.
AutoSizeMode The base size used for autoscaling the form.
AutoScroll The form has the automatic capability of scrolling.
AutoScrollMargin The size of the margin for the autoscroll.
AutoScrollMinSize The minimum size of the autoscroll.
AutoScrollPosition The location of the autoscroll position.
ClientSize The size of the client area of the form.
DefaultSize The protected property that sets the default size of the form.
DesktopBounds The size and location of the form.
DesktopLocation The location of the form.
Height The height of the form
MaximizeSize The maximum size for the form.
MinimizeSize The minimum size for the form.
Size The size of the form. set or get a Size object that contains an x, y value.
SizeGripStyle The style of the size grip used on the form.
A value from the SizeGripStyle enumerator. Values are Auto (automatically displayed when needed), Hide (hidden), or Show (always shown).

StartPosition The starting position of the form. This is a value from the FormStartPosition enumerator. Possible FormStartPosition enumeration values are CenterParent (centered within the parent form), CenterScreen (centered in the current display screen), Manual (location and size determined by starting position), WindowsDefaultBounds positioned at the default location), and WindowsDefaultLocation (positioned at the default location, with dimensions based on specified values for the size).
Width The width of the form.

Exemplul 5

```
using System.Windows.Forms;

public class FormSize : Form
{
public static void Main( string[] args )
 {
 FormSize myForm = new FormSize();
 myForm.Text = "Form Sizing";

 myForm.Width = 400;
 myForm.Height = 100;

  myForm.StartPosition =
       FormStartPosition.Manual;


 Application.Run(myForm);
 }
 }
```

Schimbarea culorilor si schimbarea fundalului unui form

Pentru a schimba culoarea de fundal a unui form se utilizeaza proprietatea BackColor. Culorile pot fi luate din structura Color aflata in spatul de nume System.Drawing.

Pentru a schimba culoarea de fundal se poate utiliza o imagine de fundal importata dintr-un fisier. O imagine este setata prin intermediul proprietatii BackgroundImage care pastreaza ca valoare un obiect de tip Image.

Clasa Image din spatiul de nume System.Drawing contine metoda statica FromFile( ) care returneaza un obiect de tip Image. Un obiect din clasa Image include proprietatile Width si Height. In program inaltimea si lungimea zonei client sunt setate in asa fel incat sa aiba aceeasi valoare ca si inaltimea si latimea imaginii.

Exemplul 6

```
using System.Windows.Forms;
using System.Drawing;

public class PicForm : Form
{
public static void Main( string[] args )
 {
 PicForm myForm = new PicForm();
 myForm.BackColor = Color.HotPink;
 myForm.Text = "PicForm - Backgrounds";

 if (args.Length >= 1)
 {
 myForm.BackgroundImage = Image.FromFile(args[0]);

 Size tmpSize = new Size();
 tmpSize.Width = myForm.BackgroundImage.Width;
 tmpSize.Height = myForm.BackgroundImage.Height;
 myForm.ClientSize = tmpSize;

 myForm.Text = "PicForm - " + args[0];
 }

 Application.Run(myForm);
 }
 }
```

Schimbarea frontierei (border) unui form

Controlarea border-ul unui form este necesara din punct de vedere estetic. De asemenea determina daca un form poate fi redimensionat sau nu. Pentru a modifica un border trebuie setata proprietatea FormBorderStyle cu una din urmatoarele enumeratii:

Fixed3D The form is fixed (not resizable) and has a 3D border.

FixedDialog The form is fixed (not resizable) and has a thick border.

FixedSingle The form is fixed (not resizable) and has a single-line border.

FixedToolWindow The form is fixed (not resizable) and has a tool window border.

None The form has no border.

Sizeable The form is resizable.

SizeableToolWindow The form has a resizable tool window border.

Exemplul 7

```
using System.Windows.Forms;
using System.Drawing;

public class BorderForm : Form
{
 public static void Main( string[] args )
 {
 BorderForm myForm = new BorderForm();
 myForm.BackColor = Color.SteelBlue;
 myForm.Text = "Borders";

 myForm.FormBorderStyle =
      FormBorderStyle.Fixed3D;

 Application.Run(myForm);
 }
 }
```

Adaugarea componentelor si control-erelor

Cuvantul *control* este folosit pentru a se referi *la obiectele de interfata* cum ar fi *butoane, bare de defilare (scroll bars), campuri de editare (labels),* etc.

Clasa Control implementeaza aspectul vizual al Form-urilor, iar fiecare obiect particular (buton, campuri de editare, etc.) implementeaza particularitati specifice si stie sa raspunda la mesajele adresate direct sau indirect.

Tehnic termenul control refera orice obiect .NET care implementeaza clasa Control. In practica, utilizam cuvantul control pentru a ne referi la orice obiect vizual gazduit de un form. Aceasta este in contrast cu o componenta care are la fel de multe caracteristici asemanatoare cu un control insa nu expune o interfata vizuala. Spre exemplu un *button* este un control, insa un *timer* este o componenta.

In mediul de programare Visual Studio.Net, control-erele, componentele cat si containerele (acestea din urma grupeaza un set de control-ere) se gasesc in Toolbox.

In exemplul urmator sunt implementate un label si un button. De asemenea, notati modul in care putem raspunde cu un anumit cod la producerea unui eveniment.

```
using System;
using System.Windows.Forms;
using System.Drawing;

public class ButtonApp : Form
{
private Label myDateLabel;
private Button btnUpdate;

public ButtonApp()
{ InitializeComponent();}

//INSEREAZA METODA InitializeComponent() AICI

protected void btnUpdate_Click( object sender, System.EventArgs e)
{ DateTime currDate =DateTime.Now ;
this.myDateLabel.Text = currDate.ToString(); }

protected void btnUpdate_MouseEnter(object sender, System.EventArgs e)
{ this.BackColor = Color.HotPink;}

protected void btnUpdate_MouseLeave(object sender, System.EventArgs e)
{ this.BackColor = Color.Blue;}

protected void myDataLabel_MouseEnter(object sender, System.EventArgs e)
{ this.BackColor = Color.Yellow;}

protected void myDataLabel_MouseLeave(object sender, System.EventArgs e)
{ this.BackColor = Color.Green; }

public static void Main( string[] args )
{ Application.Run( new ButtonApp() );   /* creaza fereastra*/      }
}
```

```
private void InitializeComponent()
 { this.Text = Environment.CommandLine;
 this.StartPosition = FormStartPosition.CenterScreen;
 this.FormBorderStyle = FormBorderStyle.Fixed3D;

 myDateLabel = new Label(); // Creaza label

 DateTime currDate ;
 currDate = DateTime.Now;
 myDateLabel.Text = currDate.ToString();

 myDateLabel.AutoSize = true;
 myDateLabel.Location = new Point( 50, 20);
 myDateLabel.BackColor = this.BackColor;

 this.Controls.Add(myDateLabel); // Adauga label-ul ferestrei

 this.Width = (myDateLabel.PreferredWidth + 100); // Seteaza latimea ferestrei pe baza latimii labelui

 btnUpdate = new Button(); // Creaza button

 btnUpdate.Text = "Update";
 btnUpdate.BackColor = Color.LightGray;
 btnUpdate.Location = new Point(((this.Width/2) -
(btnUpdate.Width / 2)), (this.Height - 75));

 this.Controls.Add(btnUpdate); // Adauga button-ul ferestrei

 btnUpdate.Click += new System.EventHandler(this.btnUpdate_Click);
 btnUpdate.MouseEnter += new System.EventHandler(this.btnUpdate_MouseEnter);
 btnUpdate.MouseLeave += new System.EventHandler(this.btnUpdate_MouseLeave);
 myDateLabel.MouseEnter += new System.EventHandler(this.myDataLabel_MouseEnter);
 myDateLabel.MouseLeave += new System.EventHandler(this.myDataLabel_MouseLeave);
 }
```

Crearea unui form utilizand Visual Studio .NET

Atunci cand creati un form ar trebui sa incercati sa utilizati urmatoarele recomandari:

1. Aliniati labelurile la dreapta form-ului si utilizati doua puncte (:) dupa text;
2. Alegeti numele controlerelor asa incat acestea sa descrie scopul lor. Utilizati un prefix pentru a indica tipul controlerului;
3. Utilizati culorile default si fonturile default;
4. Utilizati controlerul corespunzator interactivitatii pe care o doriti;
5. Aliniati butoanele OK si Cancel in coltul din dreapta jos al ferestrei;
6. Mentineti controlerele si labelurile aliniate in coloane imaginare;
7. Utilizati tabindexul pentru a naviga prin intermediul tastaturii.

Introducere in Windows Forms
Control-ere

Control-ere

Control-erele din .NET deriva din clasa System.Windows.Forms.Control Acesta clasa defineste functionalitatea control-erelor. Din acest motiv, o mare parte a proprietatilor si evenimentelor control-erelor sunt identice.

O parte a claselor derivate din clasa Control sunt la randul lor clase de baza pentru alte controlere, cum este spre exemplu cazul claselor Label si TextBoxBase, din diagrama de mai jos:

Object --- MarshalByRefObject --- Component --- Control --- Label --- LinkLabel
--- TextBoxBase --- TextBox
--- RichTextBox
--- ButtonBase --- Button
--- CheckBox
--- RadioButton
--- ListView
--- etc.

Unele controlere (numite user controls) deriva din clasa *System.Windows.Forms.UserControl* care deriva la randul ei din clasa Control si asigura functionalitatea necesara pentru a ne crea propriile noastre controlere. Controlerele utilizator utilizate pentru a crea interfete Web design deriva dintr-o alta clasa, si anume *System.Web.UI.Control.*

Clasa Control contine un numar mare de proprietati, evenimente si metode.

Proprietati ale clasei Control

Toate controlerele au nu numar mare de proprietati care sunt utilizate pentru a manipula comportarea unui anumit controler. Clasa de baza a controlerelor, System.Windows.Forms.Control, are un numar de proprietati pe care celelalte controlere le mostenesc direct, sau le redefinesc pentru a oferi comportari specifice.

Mai jos sunt trecute in revista cateva dintre proprietatile clasei Control. Pentru o descriere detaliata a tuturor proprietatilor, este indicat sa consultam documentatia .NET Framework SDK.

Nume	Descriere
Anchor	Gets or sets the edges of the container to which a control is bound and determines how a control is resized with its parent.
BackColor	Gets or sets the background color for the control.
BackgroundImage	Gets or sets the background image displayed in the control.
Dock	Gets or sets which control borders are docked to its parent control and determines how a control is resized with its parent.
Enabled	Gets or sets a value indicating whether the control can respond to user interaction.
ForeColor	Gets or sets the foreground (prim-plan) color of the control.
Name	Gets or sets the name of the control.This name can be used to reference the control in code.
TabIndex	Gets or sets the tab order of the control within its container.
TabStop	Gets or sets a value indicating whether the user can give the focus to this control using the TAB key.
Tag	Gets or sets the object that contains data about the control.This value is usually not used by the control itself and is there for you to store information about the control on the control itself. When this property is assigned a value through the Windows Form designer, you can only assign a string to it.
Text	Gets or sets the text associated with this control.
Visible	Gets or sets a value indicating whether the control and all its parent controls are displayed.

Evenimente ale clasei Control

Evenimentele generate de controlerele din Windows Forms sunt de regula asociate actiunii utilizatorului. Spre exemplu, atunci cand utilizatorul apasa cu mouse-ul pe un button, acel button genereaza un eveniment care specifica ce i s-a intamplat. Tratarea evenimentului este modalitatea prin care programatorul ofera functionalitate button-ului respectiv.

Clasa Control defineste un numar mare de evenimente comune controlerelor. Mai jos sunt trecute in revista cateva dintre evenimentele clasei Control. Pentru o descriere detaliata a tuturor evenimentelor, este indicat sa consultam documentatia .NET Framework SDK.

Nume	Descriere
Click	Occurs when the control is clicked.
DoubleClick	Occurs when the control is double-clicked. Handling the Click event on some controls, such as the Button control will mean that the DoubleClick event can never be called.
KeyDown	Occurs when a key becomes pressed while the control has focus. This event always occurs before KeyPress and KeyUp.
KeyPress	Occurs when a key becomes pressed while a control has focus. This event always occurs after KeyDown and before KeyUp. The difference between KeyDown and KeyPress is that KeyDown passes the keyboard code of the key that has been pressed, while KeyPress passes the corresponding char value for the key.
KeyUp	Occurs when a key is released while a control has focus. This event always occurs after KeyDown and KeyPress.
Validated	This event is fired when a control with the CausesValidation property set to true is about to receive focus. It fires after the Validating event finishes and indicates that validation is complete.
Validating	Fires when a control with the CausesValidation property set to true is about to receive focus. Note that the control which is to be validated is the control which is losing focus, not the one that is receiving it.

Tratarea unui eveniment

Exista trei posibilitati pentru a insera codul de tratare pentru un anumit eveniment. Primul dintre acestea incepe prin a face dublu-click pe control-erul avut in discutie. Suntem dusi la metoda care trateaza evenimentul asociat prin default control-erului. (Daca controlerul este un Button evenimentul asociat prin default este Click, daca este un TextBox atunci evenimentul este TextChanged etc.) Daca acesta este evenimentul dorit atunci codul dorit se introduce in blocul metodei care trateaza evenimentul.

Daca insa se doreste un un alt eveniment decat evenimntul default atunci sunt doua posibilitati. Una dintre acestea este de a alege evenimentul dorit din lista de evenimente a ferestrei Properties. Evenimentul scris pe fond gri este evenimentul default. Pentru a alege un alt eveniment si a adauga metoda de tratare a evenimentului se face dublu click pe evenimentul dorit din lista de evenimente. Metoda care trateaza evenimentul considerat este generata automat, numele metodei fiind foarte sugestiv. Spre exemplu, daca controlerul se numeste textBox1, evenimentul considerat este Click atunci metoda care trateaza evenimentul se va numi textBox1_Click. Altfel, puteti tasta numele dorit de dumneavoastra pentru metoda de tratare a evenimentului in dreptul evenimentului dorit, iar apoi sa apasati enter. Metoda avand numele dorit de dumneavoastra va fi generata automat.

O alta optiune este de a adauga codul care creaza evenimentul in blocul constructorului, dupa apelul metodei InitializeComponent(), iar apoi de a adauga metoda care trateaza evenimentul, bineinteles in afara blocului constructorului.

Aceste ultime doua posibilitati necesita doi pasi: crearea evenimentului si crearea metodei de tratare a evenimetului.

In cele ce urmeaza vom prezenta utilitatea catorva control-erele (Clase derivate din clasa Control a caror instante asigura o functionaltate specifica).

Clasa ButtonBase si clasele sale derivate

Atunci cand ne gandim la un Button, ne imaginam probabil un buton dreptunghiular care realizeaza anumite sarcini atunci cand facem click pe acesta. Desi aceasta este principala menire a unui button, totusi, .NET ofera o clasa derivata din clasa Control, si anume System.Windows.Foms.ButtonBase care implementeaza functionalitatile de baza necesare control-relor de tip butoane, incat orice programator poate sa isi defineasca propriile clase derivate din aceasta clasa si sa isi creeze butoanele dorite.

In fapt spatiul de nume System.Windows.Forms pune la dispozitie trei control-ere care deriva din clasa ButtonBase, si anume: Button, CheckBox and RadioButton.

Clasa Button. Acest control-er este butonul standard dreptunghiular, care exista pe orice fereastra de dialog. O instanta a clasei Button este utilizata pentru a realiza una din urmatoarele sarcini:

-inchide o fereastra de dialog (spre exemplu, butoanele OK sau Cancel)

-actioneaza asupra unor date introduse intr-o fereastra (spre exemplu butonul Search dupa ce au fost introduse anumite criterii de cautare)

-deschide o fereastra sau o aplicatie (spre exemplu, butonul Help)

Utilizarea unui button este o operatie foarte simpla. In mod uzual consta in adaugarea control-er form-ului nostru si introducerea codului de tratare a evenimentului asociat button-ului, eveniment care in majoritatea cazurilor este Click. Mai jos sunt prezentate cateva dintre proprietatile si evenimentele asociate unui button.

Proprietati (Button):

Nume	Descriere
FlatStyle	The style of the button can be changed with this property. If you set the style to Popup, the button will appear flat until the user moves the mouse pointer over it. When that happens, the button pops up to its normal 3D look.
Enabled	We'll mention this here even though it is derived from Control, because it's a very important property for a button. Setting the Enabled property to false means that the button becomes grayed out and nothing happens when you click it.

Image	Allows you to specify an image (bitmap, icon, etc.), which will be displayed on the button.
ImageAlign	With this property, you can set where the image on the button should appear.

Evenimente (Button):

Cel mai important eveniment este Click. Acesta se produce atunci cand utilizatorul face click pe button. De asemenea, evenimentul este lansat si atunci cand button-ul este focalizat (is focused) si se apasa tasta Enter.

Clasele RadioButton si CheckBox

Desi sunt derivate din aceeasi clasa ca si clasa Button, instantele acestor clase difera substantial atat prin modul in care apar intr-un form cat si prin modul lor de utilizare fata de un button.

Un radioButton, in mod traditional, apare ca un label cu un mic cerculet in stanga, care poate fi selectat sau nu. Aceste butoane (radioButtons) sunt utilizate atunci cand se doreste oferirea posibilitatii utilizatorului de a alege o optiune dintr-o multime de optiuni. Pentru a grupa mai multe radioButton-uri, se utilizeaza in prealabil un GroupBox. Se plaseaza mai intai un GroupBox in form-ul considerat, iar apoi se adauga in interiorul GroupBox-ului atatea radioButton-uri cate sunt necesare.

Un checkBox, in mod traditional, apare ca un label cu un mic dreptunghi in fata care poate avea un semn in interiorul sau. Un checkBox se utilizeaza atunci cand se doreste ca utilizatorul sa aleaga o optiune.

Proprietati (RadioButton):

Nume	Descriere
Appearance	A RadioButton can be displayed either as a label with a circular check to the left, middle or right of it, or as a standard button. When it is displayed as a button, the control will appear pressed when selected and not pressed otherwise.

AutoCheck	When this property is true, a check mark is displayed when the user clicks the radio button. When it is false the radio button must be manually checked in code from the Click event handler.
CheckAlign	By using this property, you can change the alignment of the check box portion of the radio button. The default is ContentAlignment.MiddleLeft.
Checked	Indicates the status of the control. It is true if the control has a check mark, and false otherwise.

Evenimente (RadioButton):

Nume	Descriere
CheckedChanged	This event is sent when the check of the RadioButton changes.
Click	This event is sent every time the RadioButton is clicked. This is not the same as the CheckedChange event, because clicking a RadioButton two or more times in succession only changes the Checked property once -and only if it wasn't checked already. Moreover, if the AutoCheck property of the button being clicked is false, the button will not get checked at all, and again only the Click event will be sent.

Proprietati (CheckBox):

Nume	Descriere
CheckState	Unlike the RadioButton, a CheckBox can have three states: Checked, Indeterminate, and Unchecked. When the state of the check box is Indeterminate, the control check next to the label is usually grayed, indicating that the current value of the check is not valid or has no meaning under the current circumstances.
ThreeState	When this property is false, the user will not be able to change theCheckState state to Indeterminate. You can, however, still change the CheckState property to Indeterminate from code.

Evenimente (CheckBox):

Nume	Descriere
CheckedChanged	Occurs whenever the Checked property of the check box changes. Note that in a CheckBox where the ThreeState property is true, it is possible to click the check box without changing the Checked property. This happens when the check box changes from checked to indeterminate state.
CheckedStateChanged	Occurs whenever the CheckedState property changes. As Checked and Unchecked are both possible values of the CheckedState property, this event will be sent whenever the Checked property changes. In addition to that, it will also be sent when the state changes from Checked to Indeterminate.

Clasele Label si LinkLabel

Instantele clasei Label sunt cele mai utilizate control-ere dintre toate. Aproape fiecare fereastra contine cel putin un Label. Label-ul este un control-er simplu care isi propune un singur lucru si anume sa afiseze un text pe un form.

Arhitectura .NET include doua control-ere label:

-Label, label-ul standard din Windows;

-LinkLabel, un label similar celui anterior si derivat din acesta, insa se prezinta ca un link Internet.

In mod uzual, nu este necesar nici un eveniment pentru Label-ul standard. In schimb, in cazul LinkLabel-ului este necesar cod suplimentar pentru a permite utilizatorului sa acceseze pagina de internet afisata de Text-ul linkLabel-ului.

Un numar mare de proprietati pot fi setate pentru un Label. Multe dintre acestea sunt derivate din clasa Control, insa unele sunt noi. In urmatoarea lista sunt amintite cateva dintre acestea. Daca nu este precizat altfel, proprietatile sunt comune ambelor control-ere.

Proprietati (Label si LinkLabel):

Nume	Descriere
BorderStyle	Allows you to specify the style of the border around the Label. The default is no border.
DisabledLinkColor	(LinkLabel only) The color of the LinkLabel after the user has clicked it.
FlatStyle	Controls how the control is displayed. Setting this property to Popup will make the control appear flat until the user moves the mouse pointer over the control. At that time, the control will appear raised.
Image	This property allows you to specify a single image (bitmap, icon, etc.) to be displayed in the label.
ImageAlign	(Read/Write) Where in the Label the image is shown.
LinkColor	(LinkLabel only) The color of the link.
TextAlign	Where in the control the text is shown.

Clasa TextBoxBase si clasele sale derivate

Clasa TextBoxBase ofera functionalitatea necesara utilizarii si prelucrarii unui text, precum selectarea unui text, cut si paste dintr-un si intr-un Clipbord si multe alte evenimente.

Din clasa TextBoxBase deriva doua clase, si anume TextBox si RichTextBox. Ambele control-ere preiau textul introdus de utilizator.

Un TextBox se utilizeza de regula atunci cand se doreste introducerea unui text care este necunoscut la momentul design-ului. In egala masura se pot introduce orice caractere, sau se poate forta introducerea doar a unor caractere, spre exemplu doar valori numerice.

Un RichTextBox are multe din proprietatile unui TextBox, insa prezinta si unele aspecte mai divese. Daca scopul principal al unui TextBox este acela de a obtine, de la utilizator, un text scurt, un RichTextBox se utilizeaza pentru a introduce si afisa un text sub un anumit format (spre exemplu bold, underline sau italic). Utilizeaza un format standard pentru text numit Rich Text Format sau RTF.

Amintim cateva dintre proprietatile si evenimentele claselor TextBox si RichTextBox.

Proprietati (TextBox):

Nume	Descriere
CausesValidation	When a control that has this property set to true is about to receive focus, two events are fired: Validating and Validated. You can handle these events in order to validate data in the control that is losing focus. This may cause the control never to receive focus. The related events are discussed below.
CharacterCasing	A value indicating if the TextBox changes the case of the text entered. The possible values are: l Lower: All text entered is converted lower case. 2 Normal: No changes are made to the text. 3 Upper: All text entered is converted to upper case.
MaxLength	A value that specifies the maximum length in characters of any text, entered into the TextBox.
Multiline	Indicates if this is a multiline control, which means it is able to show multiple lines of text. When Multiline property is set to true, you'll usually want to set WordWrap to true as well.
PasswordChar	Specifies if a password character should replace the actual characters entered into a single line TextBox. If the Multiline property is true then this has no effect.
ReadOnly	A Boolean indicating if the text is read only.

ScrollBars	Specifies if a multiline TextBox should display scrollbars.
SelectedText	The text that is selected in the TextBox.
SelectionLength	The number of characters selected in the text. If this value is set to be larger than the total number of characters in the text, it is reset by the control to be the total number of characters minus the value of SelectionStart.
SelectionStart	The start of the selected text in a TextBox.
WordWrap	Specifies if a multiline TextBox should automatically wrap words if a line exceeds the width of the control.

Evenimente (TextBox):

Nume	Descriere
Enter, GotFoccus Leave, Validating Validated LostFocus	These six events occur in the order they are listed here. They are known as "Focus Events" and are fired whenever a control's focus changes, with two exceptions. Validating and Validated are only fired if the control that receives focus has the CausesValidation property set to true. The reason why it's the receiving control that fires the event is that there are times where you do not want to validate the control, even if focus changes. An example of this is if the user clicks a Help button.
KeyDown, KeyPress, KeyUp	These three events are known as "Key Events". They allow you to monitor and change what is entered into your controls. KeyDown and KeyUp receive the key code corresponding to the key that was pressed. This allows you to determine if special keys such as *Shift* or *Control* and *F1* were pressed. KeyPress, on the otherhand, receives the character corresponding to a keyboard key. This means that the value for the letter "a" is not the same as the letter "A". It is useful if you want to exclude a range of characters, for example, only allowing numeric values to be entered.
TextChanged	Occurs whenever the text in the TextBox is changed, no matter what the change.

Proprietati (RichTextBox):

Nume	Descriere
Rtf	This corresponds to the Text property, except that this holds the text in RTF.
SelectedRtf	Use this property to get or set the selected text in the control, in RTF. If you copy this text to another application, for example, Word, it will retain all formatting.
SelectedText	Like SelectedRtf you can use this property to get or set the selected text. However, unlike the RTF version of the property, all formatting is lost.
SelectionAlignment	This represents the alignment of the selected text. It can be Center, Left, or Right.
SelectionBullet	Use this property to find out if the selection is formatted with a bullet in front of it, or use it to insert or remove bullets.
BulletIndent	Use this property to specify the number of pixels a bullet should be indented.
SelectionColor	Allows you to change the color of the text in the selection.
SelectionFont	Allows you to change the font of the text in the selection.
SelectionLength	Using this property, you either set or retrieve the length of a selection.
ShowSelectionMargin	If you set this property to true, a margin will be shown at the left of the RichTextBox. This will make it easier for the user to select text.
SelectionProtected	You can specify that certain parts of the text should not be changed by setting this property to true.

Evenimente (RichTextBox):

Nume	Descriere
LinkedClick	This event is sent when a user clicks on a link within the text.
Protected	This event is sent when a user attempts to modify text that has been marked as protected.
SelectionChanged	This event is sent when the selection changes. If for some reason you don't want the user to change the selection, you can prevent the change

Windows Forms

-Realizarea meniurilor

-Controlerul ToolStrip

-Aplicatii MDI si SDI

Realizarea Meniurilor

Majoritatea aplicatiilor Windows contin meniuri. Ahitectura .NET furnizeaza controlere pentru crearea rapida si simpla a meniurilor.

Controlerele MenuStrip si ContextMenuStrip

Aceste controlere, noi odata cu versiunea .NET 2.0, inlocuiesc controlerele MainMenu si ContextMenu, desi si acestea sunt pastrate pentru motive de compatibilitate. Adaugarea acestora unei ferestre se face la fel ca orice alt controler, cu deosebirea ca se plaseaza atat pe fereastra cat si in josul acesteia.

Daca controlerele MainMenu si ContextMenu se compun din obiecte de tip MenuItem, in cazul controlerelor MenuStrip si ContextMenuStrip itemii care le apartin sunt obiecte din unele clase derivate ale clasei ToolStripItem, si anume ToolStripMenuItem, ToolStripComboBox, ToolStipTextBox si ToolStripSeparator. Un obiect instanta al uneia dintre clasele enumerate mai sus poate la randul sau sa fie parinte pentru un submeniu.

Spre deosebire de meniul MenuStrip, care se plaseaza in partea de sus a ferestrei, ContextMenuStrip reprezinta un meniu de context care isi face aparitia in locul in care se face un click dreapta pe un controler (daca meniul de context a fost legat de controlerul respectiv). Este folosit de obicei pentru un acces rapid spre unii itemi ai meniului principal.

In aplicatia aferenta acestui curs vom exemplifica cum se utilizeaza proprietatilor si evenimentele acestor controlere.

Controlerul ToolStrip

Desi meniurile ofera o mare functionalitate aplicatiilor, unii itemi beneficiaza de o atentie sporita fiind plasati atat in meniu cat si in toolbar. In felul acesta se asigura acces mai rapid la unele operatii utilizate frecvent, precum Open, Save, Undo, Paste, etc.

La fel ca in cazul meniurilor, controlerul ToolStrip este nou in .NET 2.0 si inlocuieste controlerul ToolBar.

Spre exemplu, in cazul aplicatilor Word, o selectie de elemente care compun controlerul poate fi:

Un buton care apartine unui controler ToolStrip (sau ToolBar) contine de regula o imagine fara text, desi este posibil sa avem butoane care sa contina ambele elemente. In imaginea de mai sus avem butoane de primul tip. Butoane care contin text pot fi gasite spre exemplu in Internet Explorer. Daca mouse-ul este indreptat spre un buton atunci acesta afiseaza un mesaj (tooltip) care furnizeaza informatii privind scopul butonului.

Elementele unui controler ToolStrip sunt obiecte instante ale urmatoarelor clase: Button, Label, ComboBox, TextBox, Separator, ProgressBar, SplitButton, DropDownButton).

Se pot seta cateva proprietati ale controlerului, precum pozitia pe ecran, insa fiecare element (Button, Label, etc.) este independent fata de celelalte.

Controlerele SplitButton si DropDownButton contin aceleasi obiecte ca si controlerul MenuStrip (si anume ToolStripMenuItem, ToolStripComboBox, ToolStipTextBox si ToolStripSeparator). Diferenta dintre cele doua controlere, SplitButton si DropDownButton, este aceea ca atunci cand se face click pe controlerul DropDownButton sunt afisati itemii acestuia, spre deosebire de SplitButton care afiseaza itemii atunci cand se actioneaza partea din dreapta a butonului (sageata).

Aplicatii MDI si SDI

In mod traditional, sunt trei tipuri de aplicatii care pot fi programate pentru Windows. Aceste sunt:

-- aplicatii de tip dialog. Acestea se prezinta utilizatorului ca un dialog simplu care asigura functionalitatea aplicatiei. Acestea sunt de regula aplicatii mici, simple, destinate unei singure sarcini care necesita o cantitate minima de date. Un exemplu, il reprezinta Calculatorul care este incorporat in sistemul de operare.

-- aplicatii SDI (Simple Document Interfaces). Acestea se prezinta utilizatorului cu un meniu, unul sau mai multe toolbar-uri si o fereastra in care utilizatorul poate realiza unele sarcini. Aplicatiile de acest tip rezolva o sarcina specifica. Utilizatorul poate incarca un singur document in aplicatia in care lucreaza. De regula, sarcina ce urmeaza a fi realizata este complexa si implica o interactiune continua intre utilizator si aplicatie. Exemple de aplicatii SDI sunt WordPad sau Paint. Doar un singur document poate fi deschis la un moment dat.

-- aplicatii MDI (Multple Document Interfaces). Acestea se prezinta utilizatorului in aceeasi maniera ca si o aplicatie SDI, insa are abilitatea de a pastra mai multe ferestre deschise simultan. Un exemplu il reprezinta Adobe Acrobat Reader.

Construirea aplicatiilor MDI

O aplicatie MDI consta din cel putin doua ferestre. Prima fereastra este un container MDI sau fereasta principala (**MDI container**). O fereastra care poate fi afisata in interiorul containerului este numita “copil MDI” (**MDI child**).

Pentru a crea o aplicatie MDI se incepe la fel ca in cazul oricarei aplicatii windows. Pentru a schimba fereastra principala dintr-un form intr-un container MDI trebuie setata proprietatea IsMDIcontainer la true. Se observa cum background-ul ferestrei isi modifica culoarea pentru a marca faptul ca nu ar trebui sa fie plasate controlere pe aceasta, desi este posibila aceasta operatie si chiar rezonabila in unele circumstante.

Pentru a crea o fereastra **MDI child** trebuie adaugata o noua fereastra proiectului prin alegerea unui Windows Form in dialogul obtinut prin selectarea Project | Add New Item. Aceasta fereasta devine un **MDI child** daca este setata proprietatea sa MdiParent incat sa refere fereasta **MDI container**. Aceasta proprietate nu poate fi setata insa din panel-ul Properties ci trebuie setata scriind cod. Doua chestiuni mai raman de facut inainte ca aplicatia sa poata rula in modul convenit. Si anume trebuie ca fereastra **MDI container** sa stie care fereastra **MDI child** sa afiseze iar apoi sa o afiseze. Aceste chestiuni se realizeaza creand o instanta a form-ului care se doreste a fi afisat, iar pentru aceasta se apeleaza metoda Show().

Exemplu:

Daca spatiul de nume (numele proiectului se numeste WindowsFormsApplication6, form-ul **MDI container** se numeste frmContainer, iar form-ul **MDI child** are numele frmChild atunci se parcurg urmatorii pasi:

1. Se modifica constructorul clasei frmChild astfel:

```
public frmChild(WindowsFormsApplication6.frmContainer parent)
 {
    InitializeComponent();
    this.MdiParent = parent;
 }
```

2. Se modifica constructorul clasei frmContainer astfel:

```
public frmContainer()
 {
    InitializeComponent();
    WindowsFormsApplication6.frmChild child = new frmChild(this);
    child.Show();
 }
```