Controlli Windows

Il framework .NET mette a disposizione una nutrita serie di oggetti che possono essere inseriti in una Windows Form , i cosiddetti **Windows Controls** (Controlli Windows o, più semplicemente, Controlli).
Sono gli oggetti più comuni a tutte le interfacce grafiche come bottoni, checkbox, etc., ma troviamo anche controlli per la gestione di dati o per il dialogo con le periferiche.

Tutti **impacchettati in classi** con metodi e proprietà che ne permettono la personalizzazione a seconda delle esigenze, i controlli consentonto di realizzare applicazioni dall'aspetto professionale con uno sforzo minimo.
Dato il loro numero (più di 60), è impossibile analizzarli tutti in questa sede: ci limiteremo a mostrare quali sono le novità dei principali controlli del Framework 2.0, rimandando alla guida su VB.NET (ed alla guida in linea di Visual Studio) per maggiori dettagli sugli altri oggetti.

All'interno della *Casella degli strumenti* di Visual Studio 2005 (**Toolbox** in inglese), i controlli sono divisi in categorie:

- **All Windows Forms**: che visualizza in un'unica scheda tutti i controlli disponibili;
- **Common Controls**, che contiene gli oggetti tipicamente presenti in ogni finestra (Label, TextBox, Button, ListBox, ComboBox, etc.);
- **Containers**, che raggruppa i controlli che permettono di gestire il layout del form, ovvero la disposizione degli oggetti;
- **menù & Toolbars**, contenente oggetti che permettono di aggiungere barre dei menù, barre degli strumenti, barra di stato e menù contestuali all'applicazione;
- **Data**, che contiene gli strumenti che permettono di lavorare con fonti di dati esterne (come i database);
- **Components**, che raggruppa oggetti i quali consentono di interagire con il sistema operativo (per gestire le Active Directory ed il registro eventi di Windows, monitorare le modifiche al file system, etc.);
- **Printing**, che contiene gli oggetti necessari per aggiungere le funzionalità di stampa all'applicazione;

- **Dialogs**, contenente i controlli che consentono di visualizzare le finestre di dialogo comuni di Windows, come **Apri** e **Salva con nome**.

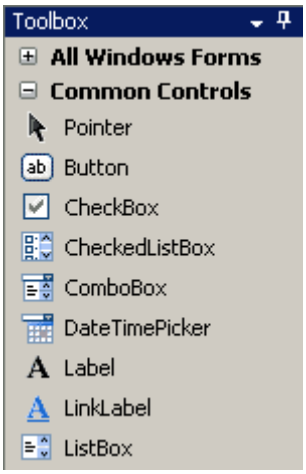

Figura 1. Toolbox di Visual Studio 2005

Controlli come TextBox, ListBox e PictureBox sono stati aggiornati con l'aggiunta di alcune proprietà e funzioni, ma il loro utilizzo è rimasto essenzialmente uguale rispetto alle versioni precedenti di .NET .

I controlli **MenuStrip** (barra dei menù) e **ToolStrip** (barra degli strumenti), sono stati notevolmente potenziati e supportano lo stile introdotto con Office 2003:

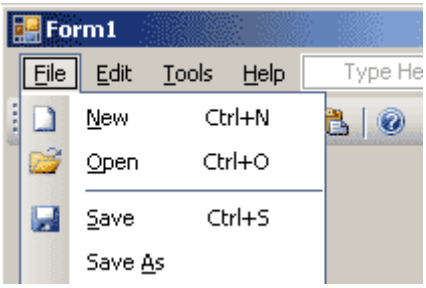

Figura 2. Nuovo stile per barra menù e strumenti

Essi sostituiscono i vecchi controlli **MainMenu** e **ToolBar**. Al loro interno è ora possibile inserire un numero maggiori di elementi: un menù, infatti, può contenere anche TextBox e ComboBox, così come la barra degli strumenti, che è in grado di ospitare anche **ProgressBar**. Il MenuStrip, inoltre, consente di associare icone alle voci di menù.

Anche il controllo **StatusStrip**, sostituto del vecchio **StatusBar** ed utilizzato per visualizzare una barra di stato nell'applicazione, ha subito le stesse operazioni di aggiornamento.
Un'altra novità è rappresentata dai controlli per gestire il **layout della finestra**, contenuti nella sezione Containers della «Casella degli strumenti». Accanto ai controlli già presenti nelle versioni del Framework, infatti, sono stati inseriti oggetti come **FlowLayoutPanel** e **tableLayoutPanel**, che consentono di disporre automaticamente i controlli inseriti al loro interno.
Mostriamo ora un semplice esempio d'uso di alcuni controlli realizzando **una piccola applicazione** per Windows: un visualizzatore di immagini minimale, composto da una casella di testo in cui digitare il percorso del file, un pulsante per visualizzare la finestra di dialogo «Apri» e un pulsante che mostra l'immagine selezionata in una PictureBox.
Una CheckBox permetterà di stabilire se si vuole visualizzare l'immagine nelle sue dimensioni reali oppure ingrandita a occupare tutta l'area disponibile.

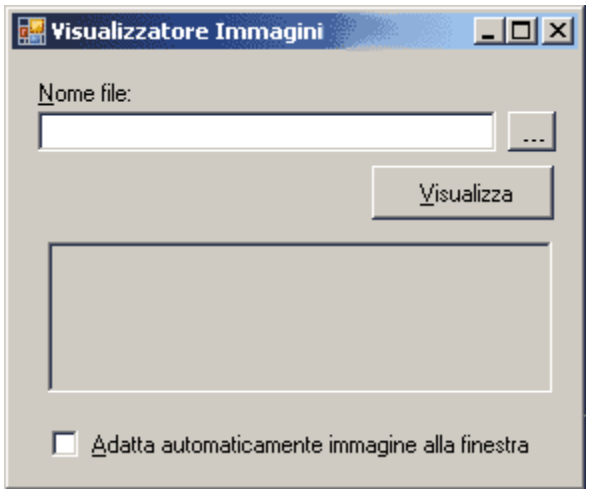

Figura 3. Interfaccia della applicazione esempio

Notiamo che è stato inserito anche un controllo di tipo **OpenFileDialog**. Ora impostiamo le proprietà dei vari oggetti come indicato nella tabella seguente:

Controllo (Classe)	Proprietà	Valore
frmVisualizzatore (Form)	AcceptButton	btnVisualizza
	StartPosition	CenterScreen
	Text	Visualizzatore Immagini
lblNomeFile (Label)	Text	&Nome file:
txtNomeFile (TextBox)		
btnSfoglia (Button)	Text	...
btnVisualizza (Button)	Text	&Visualizza
picImmagine (PictureBox)	borderStyle	Fixed3D
chkAdattaImmagine (CheckBox)	Text	&Adatta automaticamente immagine alla finestra
ofdApri (OpenFileDialog)	FileName	(Stringa vuota)
	Filter	Tutti i file immagine (*.bmp; *.jpg; *.gif; *.png)
	Title	Seleziona un file immagine

Non ci resta che scrivere il **codice da associare agli eventi** dei controlli. Anziutto vogliamo che, cliccando sul pulsante **btnSfoglia**, si apra la finestra di esplorazione per selezionare un'immagine. Per **associare automaticamente un Event Handler** all'evento principale di un determinato oggetto, è sufficiente un doppio click sull'oggetto in questione. Nel nostro caso definiamo l'event handler per l'evento «Click» del pulsante. Nella finestra del codice appare la definizione del metodo delegato e possiamo aggiungere il codice di gestione dell'evento:

```
private void btnSfoglia_Click(object sender, EventArgs e)
{
   if (ofdApri.ShowDialog() == DialogResult.OK)
       txtNomeFile.Text = ofdApri.FileName;
}
```

Il metodo ofdApri.**ShowDialog()** visualizza la finestra di dialogo per la selezione di un file; il tipo del file è indicato nella proprietà Filter del controllo.
La routine restituisce un valore di tipo **DialogResult** che permette di conoscere il motivo della chiusura della finestra: DialogResult.OK indica che la finestra è stata chiusa perché l'utente ha premuto il pulsante «OK» (se viene premuto «Annulla», il valore restituito è DialogResult.Cancel).
Il nome del file selezionato è salvato nella proprietà FileName del controllo.

Ora dobbiamo **visualizzare l'immagine** selezionata. Facciamo doppio clic sul pulsante «bntVisualizza» e scriviamo questa semplice istruzione:

```
private void btnVisualizza_Click(object sender, EventArgs e)
{
   picImmagine.Image = Bitmap.FromFile(txtNomeFile.Text);
}
```

Per caricare l'immagine, **abbiamo usato** il metodo statico FromFile() della classe Bitmap, contenuta nel namespace System.Drawing, il quale prende come argomento il nome di un file e restituisce un oggetto che rappresenta l'immagine in esso contenuta. A questo punto, è sufficiente assegnarlo alla proprietà Image del controllo picImmagine perché venga visualizzata.
Ci resta solo da definire il comportamento dell'applicazione sul click del pulsante con chkAdattaImmagine spuntata:

```
private void chkAdattaImmagine_CheckedChanged(object sender, EventArgs e)
{
   if (chkAdattaImmagine.Checked == true)
       picImmagine.sizeMode = PictureBoxsizeMode.StretchImage;
   else
       picImmagine.sizeMode = PictureBoxsizeMode.Normal;
}
```

CheckedChanged è l'evento che viene generato quando si fa clic sulla checkBox.
Queste istruzioni fanno sì che, quando la casella è selezionata, ovvero la proprietà «Checked» vale true, la proprietà «sizeMode»

della PictureBox venga imposta su **PictureBoxsizeMode.StretchImage,** in modo che l'immagine si adatta all'area disponibile.
La nostra applicazione è pronta. Possiamo eseguirla premendo il tasto F5 per verificare che il suo comportamento sia quello desiderato.