

Controlli associati ai dati

ListBox, ComboBox, DataGridView

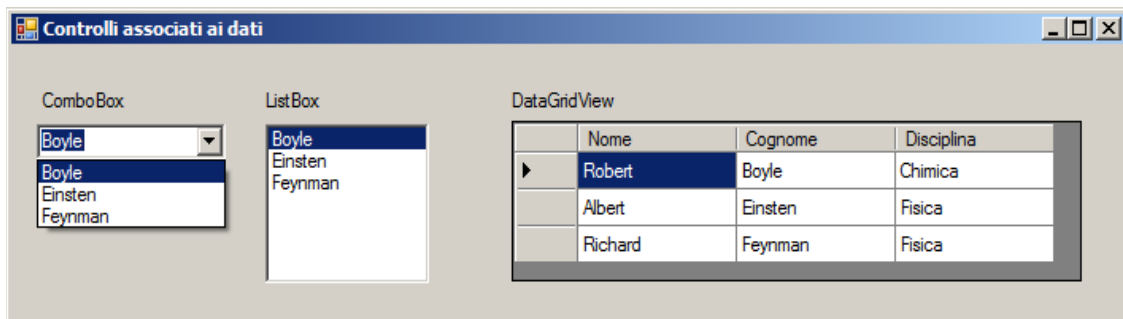
Anno 2016/2017

Indice generale

1	Introduzione.....	3
2	ListBox.....	4
2.1	“Popolare” un listbox.....	4
2.1.1	Utilizzare la collection Items del listbox.....	4
2.1.2	Associare al listbox una collezione esistente.....	4
2.2	Accesso all'elemento selezionato.....	5
2.2.1	Indice dell'elemento selezionato: SelectedIndex.....	5
2.2.2	Riferimento all'elemento selezionato: SelectedItem.....	5
2.2.3	Accesso al testo dell'elemento selezionato: proprietà Text.....	5
2.3	Impostare l'elemento selezionato.....	6
2.3.1	Impostare l'elemento selezionato mediante la proprietà Text.....	6
2.4	Gestire il “cambio di selezione”	6
3	ComboBox.....	8
3.1	Uso del ComboBox nella modalità combinata (DropDown).....	8
4	DataGridView.....	9
4.1	Design del <i>datagridview</i>	9
4.1.1	Creazione automatica delle colonne.....	9
4.2	Visualizzazione dei dati.....	9
4.3	Accesso alla riga selezionata.....	10
4.4	Gestire il cambio di selezione.....	11

1 Introduzione

Questo tutorial introduce alcuni dei “*controlli associati ai dati*”: `ListBox`, `ComboBox` e `DataGridView`. La loro caratteristica principale è quella di poter visualizzare e gestire collezioni di dati. `ListBox` e `ComboBox` si distinguono da `DataGridView`: i primi due offrono una vista di tipo monodimensionale; l'ultimo visualizza i dati in forma di griglia.



I tre controlli condividono un aspetto fondamentale: possono gestire i dati internamente, disponendo dei metodi necessari per creare, modificare ed eliminare i singoli elementi; oppure possono “appoggiarsi” a una collezione esterna. Da questa caratteristica deriva il nome di “*controlli associati ai dati*”. Infatti, con una semplice istruzione è possibile “associare” l'elenco dei dati (un vettore, un `List<>`, etc) al controllo che dovrà visualizzarli.

2 ListBox

Un *listbox* consente di visualizzare un elenco e di selezionarne un elemento. La sua funzione è quella di facilitare l'input dell'utente, permettendogli di selezionare un valore all'interno di un elenco predefinito, invece di costringerlo a digitarlo da tastiera.

2.1 “Popolare” un listbox

“Popolare” un *listbox* significa caricare l'elenco degli elementi da visualizzare. Ciò può essere fatto in vari modi:

1. a *design-time*: impostando la proprietà `Items` nel *property editor*.
2. utilizzando i metodi della proprietà `Items` del *listbox*;
3. associando la collezione degli elementi alla proprietà `DataSource`.

2.1.1 Utilizzare la proprietà `Items`

La proprietà `Items` restituisce l'elenco degli elementi visualizzati. È una collezione e definisce i metodi `Clear()`, `Add()`, `Remove()`, `IndexOf()`, etc. Il seguente codice popola il *listbox* con gli elementi del vettore `elencoCittà`:

```
string[] elencoCittà = {"Roma", "Milano", "Firenze", "Napoli", "Palermo"};
lboCittà.Items.Clear();
foreach(string città in elencoCittà)
{
    lboCittà.Items.Add(città);
}
```

Lo stesso risultato può essere ottenuto più concisamente usando `AddRange()`:

```
string[] elencoCittà = {"Roma", "Milano", "Firenze", "Napoli", "Palermo"};
lboCittà.Items.Clear();
lboCittà.Items.AddRange(elencoCittà);
```

Nota bene: l'invocazione del metodo `Clear()` garantisce che il *listbox* sia vuoto prima di essere popolato.

2.1.2 Associare al *listbox* una collezione esistente

Il modo più semplice per visualizzare un elenco di elementi è quello di “associarlo” al *listbox* utilizzando la proprietà `DataSource`.

```
string[] elencoCittà = {"Roma", "Milano", "Firenze", "Napoli", "Palermo"};
lboCittà.Items.Clear();
lboCittà.DataSource = elencoCittà
```

Nota bene: l'uso di questa tecnica impedisce di modificare successivamente l'elenco, aggiungendo o rimuovendo degli elementi.

2.2 Accesso all'elemento selezionato

Il *listbox* è un controllo di input e non di semplice visualizzazione. Nella maggior parte dei casi è necessario accedere all'elemento selezionato dall'utente. A questo scopo, il controllo definisce vari meccanismi¹.

2.2.1 Indice dell'elemento selezionato: *SelectedIndex*

La proprietà `SelectedIndex` restituisce l'indice dell'elemento correntemente selezionato. Se non c'è un elemento selezionato, la proprietà restituisce -1.

Partendo dal codice precedente e presupponendo che `elencoCittà` sia una variabile globale, è possibile visualizzare il codice della città selezionata nel *listbox*:

```
private void btnVisualizza_Click(object sender, EventArgs e)
{
    int indCittà = lboCittà.SelectedIndex;
    if (indCittà == -1) // verificare sempre che sia selezionato un elemento!
        return;

    MessageBox.Show(elencoCittà[indCittà]);
}
```

2.2.2 Riferimento all'elemento selezionato: *SelectedItem*

Diversamente da `SelectedIndex`, la proprietà `SelectedItem` fornisce un accesso diretto all'elemento selezionato, oppure `null` se non c'è nessun elemento selezionato. Per usare correttamente la proprietà è necessario specificare il tipo dell'elemento, mediante l'operatore di cast.

```
private void btnVisualizza_Click(object sender, EventArgs e)
{
    string città = (string) lboCittà.SelectedItem;
    if (città == null) // verificare sempre che sia selezionato un elemento!
        return;

    MessageBox.Show(città);
}
```

Nota bene: in entrambi gli esempi verifico che sia stato selezionato un elemento.

2.2.3 Accesso al testo dell'elemento selezionato: proprietà *Text*

Se tutto ciò che serve è conoscere il testo dell'elemento selezionato è sufficiente utilizzare la proprietà `Text`:

```
private void btnVisualizza_Click(object sender, EventArgs e)
{
    string città = lboCittà.Text; // -> "" se non è selezionato un elemento
    MessageBox.Show(città);
}
```

¹ Il *listbox* gestisce anche la selezione multipla.

Text vs SelectedItem

Le due proprietà appaiono del tutto simili, ma non lo sono. Possono essere utilizzate in modo intercambiabile se il *listbox* visualizza un elenco di stringhe.

2.3 Impostare l'elemento selezionato

Attraverso `SelectedIndex` è possibile impostare da programma l'elemento selezionato. Ad esempio, il seguente codice imposta come elemento selezionato il primo (se esiste):

```
lboCittà.SelectedIndex = 0; // produce un errore se il listBox è vuoto!
```

Il seguente codice, invece, seleziona l'ultimo:

```
lboCittà.SelectedIndex = lboCittà.Items.Count-1;
```

Infine, è possibile "deselezionare" l'elemento:

```
lboCittà.SelectedIndex = -1;
```

2.3.1 Impostare l'elemento selezionato mediante la proprietà Text

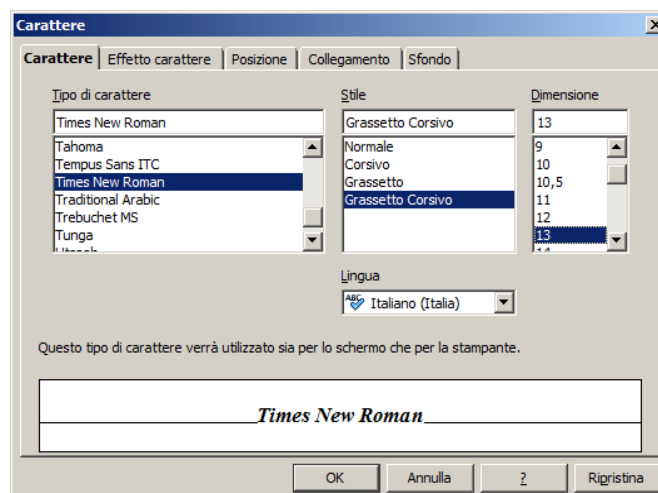
Nel caso il *listbox* visualizzi un elenco di stringhe, è possibile selezionare un elemento assegnando il suo valore alla proprietà `Text`. Ad esempio, la seguente istruzione seleziona il secondo elemento:

```
lboCittà.Text = "Milano";
```

Se il valore specificato non esiste all'interno del *listbox*, la proprietà `SelectedIndex` viene impostata a -1.

2.4 Gestire il "cambio di selezione"

Spesso, popolare il *listbox* e accedere all'elemento selezionato è tutto ciò che serve, ma non sempre è così. Consideriamo la finestra di gestione del formato carattere di un *wordprocessor*. Questa presenta tre *listbox*, che consentono di impostare il font, lo stile e la dimensione del carattere. Ebbene il cambio di selezione in ognuno dei tre *listbox* viene gestito immediatamente e si riflette sugli altri due, come si riflette sull'anteprima, che mostra il formato del carattere sulla base delle attuali impostazioni.



Per implementare una funzionalità simile è necessario gestire l'evento `SelectedIndexChanged`. Questo si

verifica ogni qual volta cambia l'indice dell'elemento selezionato. Il seguente codice gestisce il cambio della selezione di `lboCittà`, visualizzando mediante una *label* la città selezionata:

```
private void lboCittà_SelectedIndexChanged(object sender, EventArgs e)
{
    int indCittà = lboCittà.SelectedIndex;
    if (indCittà > -1)
        lblCittàSelezionata.Text = elencoCittà[indCittà];
    else
        lblCittàSelezionata.Text = "N.D.";
}
```

Nota bene: l'evento viene sollevato ogni qualvolta varia il valore di `SelectedIndex`. Ciò può avvenire in risposta alle azioni dell'utente, ma anche modificando direttamente `SelectedIndex`. Per questo motivo è sempre opportuno gestire questo evento e non l'evento `Click`. Anche quest'ultimo può implicare un cambio di selezione, ma soltanto come risultato dell'azione dell'utente.

3 ComboBox

Il *combobox* combina le funzionalità di un *textbox* e un *listbox*, ereditando da entrambi le sue caratteristiche. Tutto ciò che è stato detto sul *listbox* vale anche per il *combobox*. Quest'ultimo ha una proprietà, `DropDownStyle`, che consente di stabilire come debba comportarsi:

DropDownStyle	Descrizione
Simple	Funziona come un <i>textbox</i> .
DropDown	Funziona come <i>textbox</i> , accettando l'input da tastiera, e come <i>listbox</i> , gestendo un elenco di elementi.
DropDownList	Funziona come un <i>listbox</i> , limitandosi a gestire un elenco di elementi. Non accetta l'input da tastiera.

Rispetto al *listbox* non consente la selezione multipla e gestisce l'elenco degli elementi in una "finestra a scomparsa".

3.1 Uso del ComboBox nella modalità combinata (DropDown)

Nella maggior parte dei casi, è opportuno impostare il *combobox* nella modalità `DropDownList`, ma in alcuni scenari, può essere utile impostarlo nella modalità combinata, per consentire all'utente di inserire direttamente il valore. Il caso tipico è quello di inserimento di un sigla di provincia. Data la lunghezza dell'elenco e la brevità del dato da inserire, conviene fornire all'utente la possibilità di scegliere se digitare la sigla o selezionarla dall'elenco

In un scenario simile, è facile verificare che la sigla inserita da tastiera sia corretta. Anche il *combobox*, come il *listbox*, consente di selezionare un elemento assegnandone il valore alla proprietà `Text`. Dunque, se l'utente inserisce una sigla presente nell'elenco, sarà questa la sigla selezionata. Se, al contrario, l'utente inserisce una sigla inesistente, `SelectedIndex` restituirà -1.

4 DataGridView

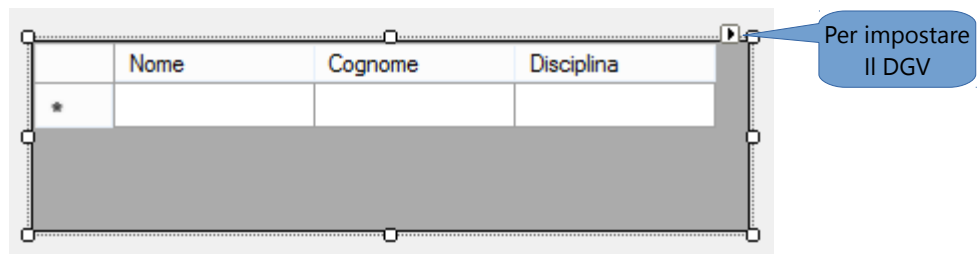
Il *datagridview* (d'ora in avanti: DGV) visualizza i dati in forma tabellare. È un controllo molto sofisticato, che include anche funzionalità di modifica e ordinamento dei dati.

4.1 Design del *datagridview*

Benché non sia obbligatorio, è opportuno che la struttura e il formato delle colonne del DGV siano stabilite a *design-time*. In questa fase è possibile, tra le altre cose:

- Creare/eliminare/modificare le colonne della griglia.
- Stabilire l'intestazione di ogni colonna.
- Stabilire il tipo di colonna: *testo*, *bottone*, *immagine*, *link*, *lista*, etc.
- Stabilire le loro caratteristiche visuali: dimensione, colore, allineamento, formattazione, etc.

È possibile impostare il DGV utilizzando il *property editor*, oppure cliccando in alto a destra sull'icona a freccia:



4.1.1 Creazione automatica delle colonne

Il DGV definisce una proprietà, `AutoGenerateColumns`, che stabilisce se debba creare automaticamente le colonne in base alla struttura dei dati. In questo tutorial parto dal presupposto che le colonne siano sempre create a *design-time*; dunque, `AutoGenerateColumns` sarà sempre impostata a *false*.

4.2 Visualizzazione dei dati

Come per i controlli *listbox* e *combobox*, esistono due modalità; la più utilizzata prevede l'uso di `DataSource`. Qui mostro l'altra, che implica la visualizzazione dei dati mediante il loro inserimento nella collezione `Rows`; quest'ultima rappresenta l'omologa della proprietà `Items` di *listbox* e *combobox*.

Si supponga di avere un elenco di record di tipo `Scienziato`:

```
public class Scienziato
{
    public string Nominativo;
    public string Disciplina;
    public DateTime DataNascita;
    public DateTime DataMorte;
}
...
List<Scienziato> scienziati = new List<Scienziato>();
...
```

```

void CreaElencoScienziati()
{
    var s = new Scienziato
    {
        Nominativo = "Curie, Maria",
        Disciplina = "Chimica",
        DataNascita = new DateTime(1867, 11, 7),
        DataMorte = new DateTime(1934, 7, 4)
    };
    scienziati.Add(s);
    ...
}

```

Si vuole visualizzare i dati su una griglia di quattro colonne. Per farlo occorre aggiungere una riga per ogni elemento nell'elenco:

```

dgv.AutoGenerateColumns = false;

dgv.Rows.Clear();
foreach (var s in scienziati)
{
    dgv.Rows.Add(s.Nominativo, s.Disciplina, s.DataNascita, s.DataMorte);
}

```

Nota bene: il metodo `Add()` consente di specificare tanti argomenti quante sono le colonne del DGV.

	Nominativo	Disciplina	Nascita	Morte
▶	Curie, Maria	Chimica	07/11/1867 00.0...	04/07/1934 00.0...
	Einstein, Albert	Fisica	14/03/1879 00.0...	18/04/1955 00.0...
*				

4.3 Accesso alla riga selezionata

Il DGV definisce una proprietà, `SelectedRows`, che restituisce una lista delle righe selezionate. (La lista è vuota se non è stata selezionata nessuna riga). Ogni elemento della lista è di tipo `DataGridViewRow` e definisce la proprietà `Index`, che rappresenta l'indice della riga nell'elenco.


Partendo dall'esempio precedente, supponiamo di voler conoscere lo scienziato selezionato:

```

private void btnVisualizza_Click(object sender, EventArgs e)
{
    if (dgv.SelectedRows.Count == 0) //nessuna riga selezionata
        return;

    int indice = dgv.SelectedRows[0].Index; //trova l'indice della prima riga selezionata
    lblScienziato.Text = scienziati[indice].Nominativo;
}

```

	Nominativo	Disciplina	Nascita	Morte
	Curie, Maria	Chimica	07/11/1867 00.0...	04/07/1934 00.0...
	Einstein, Albert	Fisica	14/03/1879 00.0...	18/04/1955 00.0...
*				

Visualizza
scienziato

Einstein, Albert

Il DGV ammette varie modalità di selezione; per selezionare un'intera riga, occorre cliccare sull'intestazione di riga corrispondente (vedi freccia rossa).

4.4 Gestire il cambio di selezione

Analogamente ai controlli *listbox* e *combobox*, anche il DGV definisce un evento che permette di gestire il cambio di selezione: `SelectionChanged`.

```
void dgv_SelectionChanged(object sender, EventArgs e)
{
    if (dgv.SelectedRows.Count == 0)
        lblScienziato.Text = "N.D.";
    else
    {
        int indice = dgv.SelectedRows[0].Index;
        lblScienziato.Text = scienziati[indice].Nominativo;
    }
}
```