di Francesco Petroni

Visual Basic Intermedio
Applicazioni SDI, MDI e Explorer Style

Siamo arrivati al terzo di una serie di articoli (cinque o forse più di cinque) dedicati al Visual Basic. Nel primo abbiamo parlato di griglie ed affini (oggetti "visibili" che ospitano insiemi di dati), nel secondo di array e collezioni (elementi "non visibili" che ospitano anch'essi insiemi di dati). In questo terzo articolo parliamo della distribuzione di un'applicazione su più form. Nel prossimo parleremo di input/output di file di vario genere (file con dati testuali e file con dati grafici), nel successivo parleremo dei rapporti tra il Visual Basic e i database, ecc. ecc.

Ripetiamo, per chiarire il taglio degli articoli, quanto già detto in una puntata precedente. Si tratta di una serie di articoli dedicati agli utilizzatori del Visual Basic, articoli di "taglio intermedio", pensati per una precisa categoria di utilizzatori, non quelli alle prime armi né quelli già esperti, ma quelli che stanno in mezzo, che hanno già le conoscenze di base e che vogliono approfondire solo alcuni temi particolari. Gli esperti invece già dovrebbero conoscere gli argomenti trattati e quindi dovrebbero anche essere in grado di svolgere facilmente gli esercizi che corredano l'articolo... ma non è detto.

Gli articoli sono prevalentemente pratici, nel senso che propongono una serie di esercizi facilmente rieseguibili da ciascuno di voi. Anche in questo terzo articolo ogni esercizio corrisponde ad una figura che mostra sia il listato sia, in primo piano, il form che costituisce l'aspetto esteriore dell'applicazione.

Terza parte

Uno, due... tanti form

Una semplicissima applicazione Visual Basic potrebbe utilizzare un solo form, che si apre al lancio dell'applicazione e si chiude con il termine della stessa.

Se l'applicazione richiede due o più form nasce il problema di come gestirli e quindi di quando e come aprirli e chiuderli.

Prima decisione fondamentale è se utilizzare un'architettura SDI, in cui ogni form va per conto suo, oppure un'architettura MDI, in cui c'è un form che fa da cornice per tutti gli altri form che via via vengono aperti e chiusi. C'è anche la variante Explorer Style, in cui assumono molta importanza gli elementi interni al form, che sono solitamente due, un controllo TreeView ed uno ListView.

Per ora ci riferiremo alle applicazioni di tipo SDI, in cui i form vivono indipendentemente gli uni dagli altri.

In un'applicazione che prevede più form normalmente esiste uno StartUp form, ovvero il form che si apre quando si lancia l'applicazione; gli altri form vengono via via aperti e chiusi al verificarsi di certi eventi. L'istruzione per far apparire un form, supponendo che si chiami F1, è la seguente:

```
Load F1
  ' carica ma non visualizza il form
F1.Show
  ' metodo che carica, se non già caricato, il form e comunque lo visualizza
```

Viceversa, per chiuderlo, quando non serve più:

```
UnLoad F1
  ' scarica il form
F1.Hide
  ' metodo che lo nasconde senza scaricarlo
```

E' evidente, dall'esame di queste semplici istruzioni, la differenza tra caricare e visualizzare un form. Caricare senza ancora visualizzare potrebbe servire per "guadagnare tempo", scaricare invece di nascondere potrebbe servire per "guadagnare memoria". In realtà neanche i manuali più approfonditi danno indicazioni su come conviene comportarsi nelle varie situazioni, occorre sperimentare e poi decidere.

Oltre al caricamento esplicito è possibile un caricamento implicito che viene eseguito quando si referenzia un controllo di un form non caricato:

```
F2.T1 = "Ciao"
```

Anche se il form F2, in cui c'è la TextBox T2, non è caricato, l'istruzione lavora correttamente.

Importantissima è la modalità di caricamento del secondo form, che può es-

sere *modal* oppure *modeless*. Nel primo caso il programma si blocca (non vengono eseguite le istruzioni successive, né si può passare agli altri form) fin quando non viene chiuso il secondo form:

```
F2.Show 1
   ' form modal: il programma si blocca
MsgBox "Sbloccato"
   ' istruzione eseguita quando F2 viene
   chiuso
```

Oppure, nel secondo caso:

```
F2.Show 0
   ' form modeless: il programma non si
   blocca
MsgBox "Sbloccato"
   ' istruzione eseguita immediatamente
```

Quindi un form modale obbliga l'utilizzatore a terminare le operazioni sul form stesso senza poter "cliccare" su elementi esterni, come altri form della stessa applicazione.

Altro problema tipico del lavoro con più form consiste nelle modalità di passaggio dati da un form ad un altro. Si possono sempre, come appena detto, referenziare controlli presenti anche in form non caricati o non visualizzati:

```
F1.T1 = "Buongiorno"
F2.T1 = "Buonanotte"
```

Sono istruzioni che funzionano sempre, da qualsiasi routine di qualsiasi form siano richiamati.

Se il passaggio dei dati tra form avviene attraverso variabili, occorre dichiararle in un modulo BAS.

```
Sub C1_Click()    ' sul primo form
  X=123
  F2.Show
End Sub

Sub F2_Load()
  T1 = X
     ' restituisce 123 se X è stata dichia-
     rata nel modulo BAS, altrimenti 0
End Sub

Global X
   ' dichiarazione da inserire nel modulo
   BAS
```

I nostri primi esercizi

Mettiamo subito a frutto quanto teorizzato fino ad ora realizzando tre semplici esercizi.

Il primo è documentato dalla figura 3 e mostra come far apparire un form che contenga una ProgressBar in movimento (effetto dinamico che non viene reso dalla figura). L'avanzamento del valore della ProgressBar, che si ottiene impostandone la proprietà Value, è ottenuto da un controllo timer che ogni secondo incrementa un contatore. L'avanzamento dura 15 secondi, dopo di che ricomincia.

Il secondo esercizio, documentato dalla figura 4, mostra un paio di form che facilitano la digitazione di quattro campi presenti in un form principale. In sostanza quando nel form principale si attiva un campo in cui digitare una data appare un form che presenta un calendario (controllo Calendar.Ocx) che facilita l'operazione in quanto mostra anni, mesi, giorni della settimana, e che permette di cliccare anziché digitare. Discorso pressoché analogo anche per la digitazione di una città da scegliere in una lista (controllo ListBox). L'unica difficoltà, in un'applicazione che usa finestre d'appoggio, è quella di "azzeccare" gli eventi più idonei all'apertura e alla chiusura dei form d'appoggio.

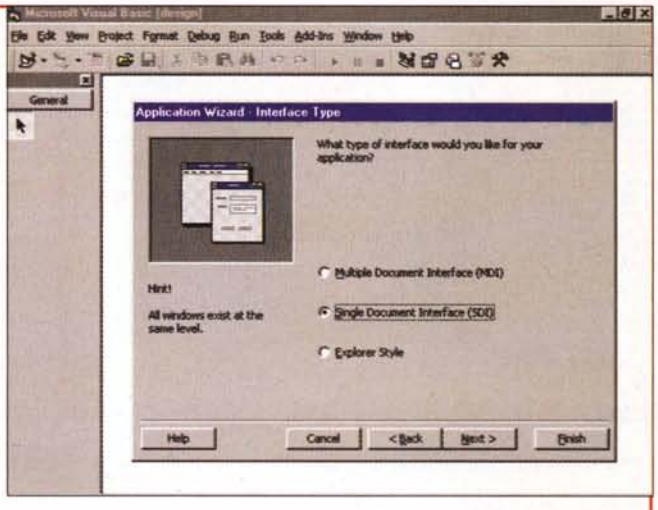


Figura 1 - Visual Basic 5.0 - SDI, MDI o Explorer Style. Non solo chiunque programmi con un qualsiasi strumento che consenta di sviluppare applicazioni per Windows, ma anche chiunque sia un semplice utilizzatore di Windows dovrebbe conoscere le varie tipologie possibili per le applicazioni Windows. SDI, quella con più finestre indipendenti, MDI, in cui tutte le finestre vivono all'interno di un form/finestra MDI, la "madre di tutte le finestre", che svolge il compito di cornice. L'ultima moda è la finestra Explorer Style (adottata ora anche dal sistema di help di Windows 98), con un controllo TreeView a sinistra ed un controllo ListView a destra. Se si lancia l'autocomposizione delle applicazioni di Visual Basic 5.0 la prima domanda riguarda proprio il tipo di applicazione desiderata e la scelta è tra le tre citate.

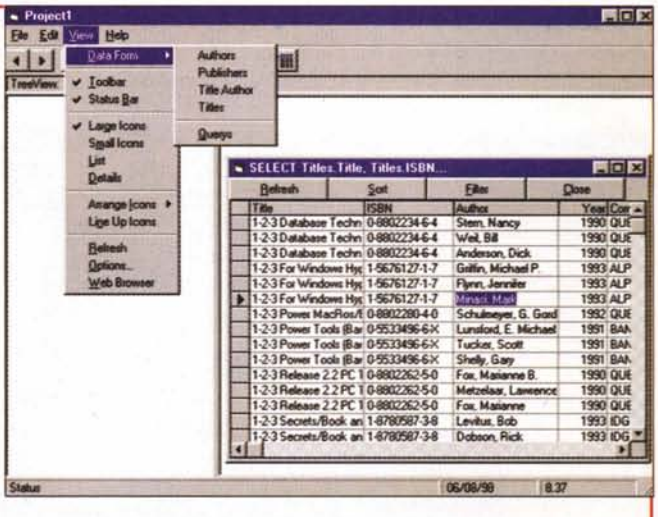


Figura 2 - Visual Basic 5.0 - A proposito di autocomposizione. Usare una procedura di autocomposizione, che risolve ovviamente solo problemi generali, per realizzare una propria applicazione, che invece deve risolvere problemi particolari, sembrerebbe una contraddizione. Invece la procedura di autocomposizione di Visual Basic 5.0 produce form "finiti" di vario tipo, pieni di oggetti (menu, toolbar, griglie, browser per Internet, ecc.) e con il codice, creato automaticamente, necessario al suo funzionamento. Il programmatore deve solo adattare il "materiale" prodotto dal wizard alle proprie particolari necessità, modificare oppure inserire il proprio codice al posto dei "segnaposto" lasciati e segnalati dalla procedura automatica. Qui vediamo il risultato di una procedura di autocomposizione incaricata di produrre un'applicazione Explorer Style.

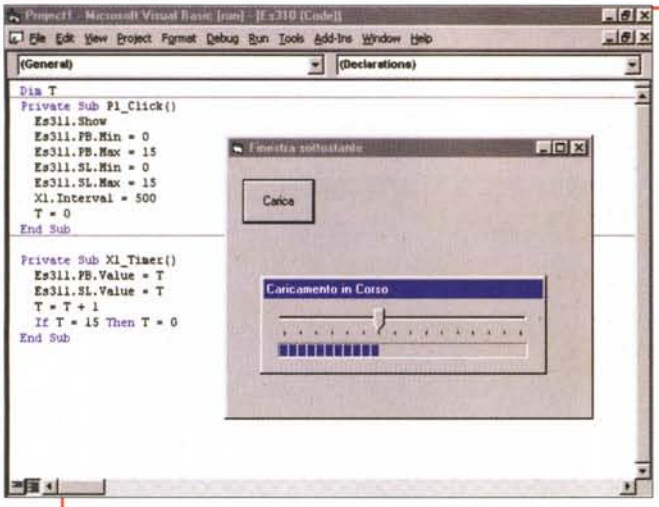


Figura 3 - Visual Basic 5.0 - Una semplice applicazione di tipo SDI con due finestre. Il nostro primo esercizio. Il primo dei due form/finestre, quello in secondo piano, contiene solo un pulsante il cui compito è quello di attivare un timer e di mostrare il secondo form, che contiene un controllo ProgressBar che avanza di "una tacca" ogni secondo. Come evidente dal listato, l'avanzamento delle ProgressBar dura quindici secondi ed alla fine di tale periodo l'avanzamento ricomincia. Il metodo Show, che si esegue su un oggetto form, permette due varianti a seconda che il form si apra come modal o modeless. Un form si dice modale (spesso tradotto in finestra a scelta obbligatoria) quando viene impedita qualsiasi operazione sugli altri form aperti al momento fin quando il form stesso non venga chiuso.

Il terzo esercizio è un'applicazione pratica di quanto visto nel primo. La ProgressBar viene fatta avanzare via via che vengono letti i dati da un database e via via che questi dati riempiono un griglia.

Per meglio capire il funzionamento del programma, documentato in figura 6, è bene conoscere la struttura del database che utilizzeremo. Lo vediamo e descriviamo in figura 5.

La sincronizzazione tra avanzamento della lettura dei dati del database e della ProgressBar è ottenuta sfruttando le proprietà RecordCount (che indica il numero totale dei record) e AbsolutePosition (numero progressivo del record corrente), gentilmente messe a disposizione da un RecordSet DAO.

Figura 4 - Visual Basic 5.0 - Un'applicazione di tipo SDI con due finestre d'appoggio. In questo secondo esempio usiamo un form principale nel quale occorre inserire quattro dati, due date e due nomi di città prelevati da una semplice lista. L'inserimento viene facilitato da due finestre SDI "volanti", la prima che mostra il controllo Calendar (il più classico degli ActiveX) e la seconda che mostra un controllo ListBox (il più classico dei controlli standard). In un'applicazione SDI tutti i form se ne vanno per i fatti propri, quindi il compito del programmatore è quello di gestirne l'apertura e la chiusura, individuando gli eventi in seguito ai quali aprirli o chiuderli.

Due o tre cose sull'architettura MDI

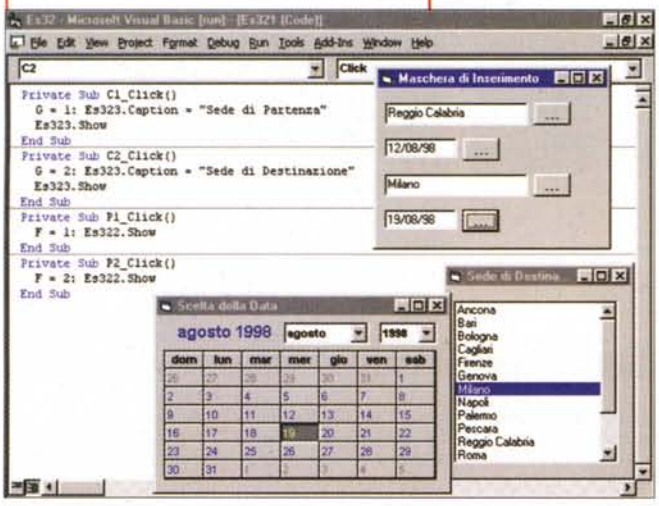
Un'applicazione MDI (Multiple Document Interface) consiste in un form (MDIForm), che fa da cornice, e da altri form (MDIChild), che vivono all'interno della cornice.

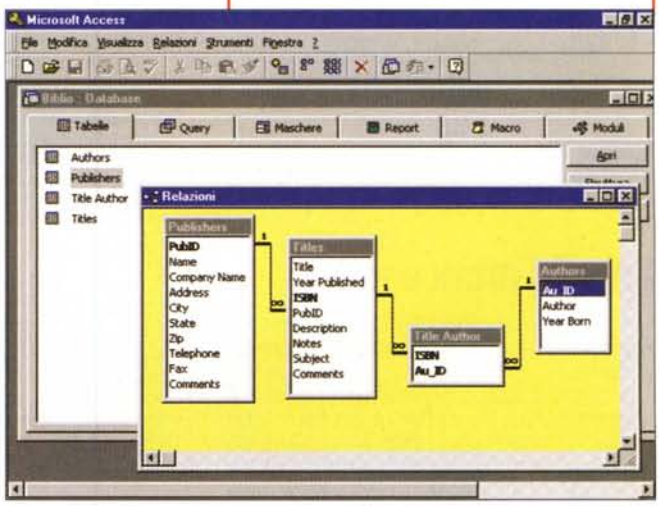
Tutti i prodotti della suite Office e conseguentemente tutti i prodotti Office-compatibles rispettano la tecnologia MDI. Ad esempio quando si usa Word e non si apre o si crea nessun documento è attiva solo la cornice MDI. Ogni documento che si apre o si crea occupa un form MDIChild.

Ovviamente, se i documenti aperti sono più di uno diventa importante la voce del menu Finestre, che consente di passare da una finestra aperta all'altra, anche se la seconda non è visibile, che consente di posizionare "in cascata" le varie finestre attive, oppure di vederle allineate in orizzontale oppure in verticale. Infine, se queste finestre sono iconizzate e sparse nella finestra cornice si può attivare la voce Allinea le Icone.

Per realizzare con il Visual Basic (qualsiasi versione) un'applicazione MDI, occorre eseguire alcuni passi obbligati (ci riferiamo al VB 5.0 Enterprise Edition versione 5.0):

Figura 5 - Visual Basic 5.0 - Applicazione d'esempio Biblio.mdb. A supporto del materiale d'esempio del prodotto MS Visual Basic 5.0 sono tradizionalmente presenti due database in formato MDB (detto anche formato Access). Si tratta dell'intramontabile file NorthWind.mdb, di argomento culinario, e del file Biblio.mdb, che contiene editori, titoli ed autori (Publishers, Titles, Authors) relativi a libri riguardanti l'informatica. Nella struttura del database è presente anche la tabella Title_Author, che serve per risolvere la problematica relazionale "molti a molti", dovuta al fatto che un titolo può essere scritto da più autori (nei testi riguardanti l'Informatica non è un caso raro). Poiché si tratta di un archivio reale, la Microsoft provvede a tenerlo aggiornato ed a metterlo a disposizione per il download dal suo sito Internet.





New Standard Exe
iniziamo un nuovo progetto
chiamiamo il primo form F1
menu Project Add New form
aggiungiamo un nuovo form
chiamiamo il secondo form F2
menu Project Add MDI form
aggiungiamo il form MDI
chiamiamo il form MDI F0
proprietà MDIChild di F1
poniamo a True la proprietà MDIChild del form F1
proprietà MDIChild di F2
poniamo a True la proprietà MDIChild del form F2
menu Project, Project Properties, StartUp Object
impostiamo F0 come oggetto di partenza

In questo modo abbiamo realizzato un'applicazione MDI composta da una cornice MDI e da due form, F1 e F2. Il problema ora è quello di gestire l'apertura e la chiusura dei due form che vivono all'interno di F0.

Normalmente conviene creare un menu nella F0, in cui ci siano almeno due rami, il primo che serve per aprire e chiudere i vari form figli (F1.Show e F2.Show, per intenderci) ed il secondo che permetta le classiche operazioni tra le finestre/form aperte.

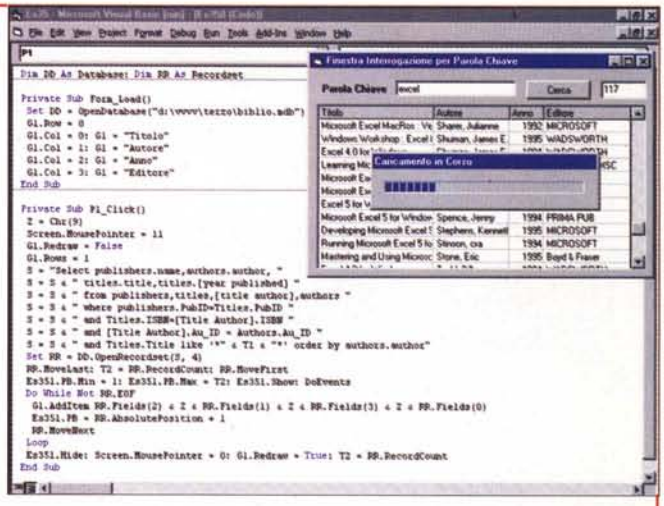


Figura 6 - Visual Basic 5.0 - Un'applicazione in due finestre - Variante con caricamento dati.
Si tratta di una variante dell'esercizio visto in figura 3, trasportato in un caso reale. Supponiamo che nella nostra applicazione ci sia un form che consenta di eseguire una ricerca per parola chiave, ad esempio una parola chiave presente nel titolo di un libro registrato nel nostro database di prova Biblio. I risultati della ricerca, ovvero l'elenco dei libri, li riversiamo in una griglia. Se il database è di dimensioni medio-grandi (superiore a diverse decine di migliaia di record) i tempi dell'operazione possono essere lunghi (da dieci secondi in su). E' opportuno e facile visualizzare un form che mostra un controllo ProgressBar che segnala l'avanzamento della ricerca. Poiché il risultato della ricerca è un RecordSet DAO se ne possono facilmente usare le due proprietà RecordCount e AbsolutePosition per pilotare la barra graduata.

Figura 7 - Visual Basic 5.0 - Esercizio base per un'applicazione MDI.
Il principio che sta alla base della tecnologia MDI è che con una stessa applicazione è normalmente possibile generare e gestire più documenti insieme. L'applicazione MDI prevede un form contenitore ed i vari documenti realizzati sono form in esso contenuti. In un'applicazione MDI c'è un solo form MDI (è un tipo particolare di form) e tanti form MDIChid. Impostando a True la proprietà MDIChild dei form figli si crea la semplice dipendenza gerarchica tra contenitore e contenuto. Tra form MDI e form MDIChild intercorrono una dozzina di regolette che occorre conoscere. Il nostro esercizio, descritto nel testo, serve per capire i meccanismi di base di un'applicazione MDI.

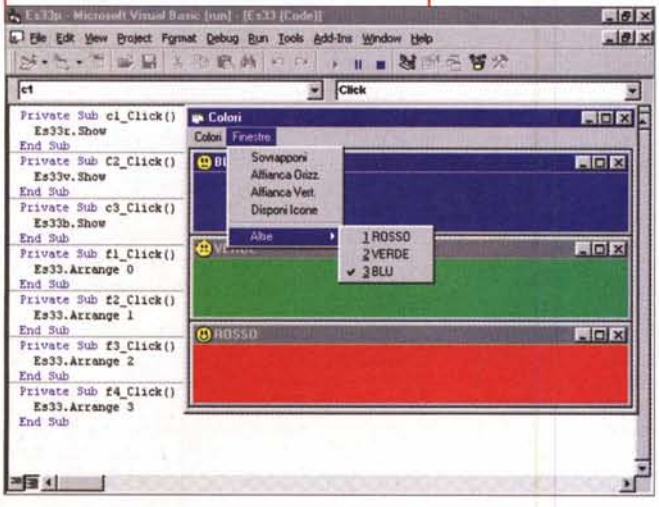
Nell'esercizio che vedete nelle figure 7 e 8 vi proponiamo un'applicazione con una cornice e tre form MDIChild contraddistinti da colori (es. proprietà Name:Rosso, Caption:Rosso, BackColor: Rosso). Il form MDI cornice, che si chiama Colori, dispone di un menu che permette di eseguire tutte le operazioni necessarie all'apertura ed alla dislocazione delle finestre.

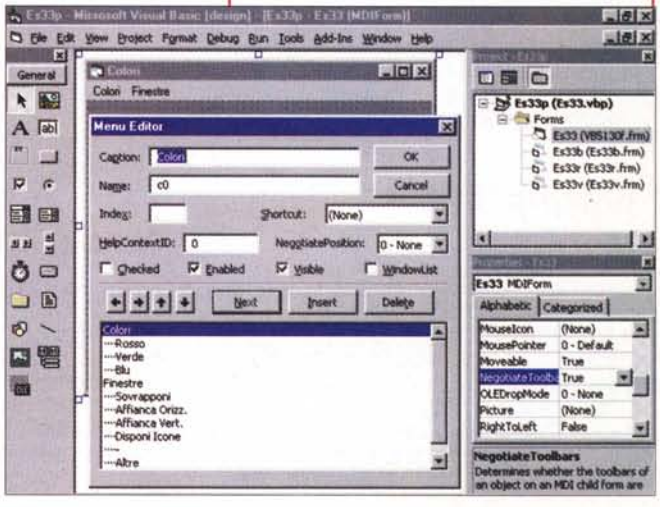
E' chiaro che quest'esercizio serve solo per comprendere il meccanismo di gestione delle finestre che è del tutto indipendente ed indifferente dal contenuto della singola finestra.

Nell'esercizio di figura 9 vediamo come sia possibile generare dinamicamente nuove finestre partendo da una finestra prototipo.

Dim FF As New Es341

Figura 8 - Visual Basic 5.0 - Il menu Finestre.
Un'applicazione di tipo MDI dispone sempre della voce di menu Finestre che comprende una serie di voci standard (affianca, sovrapponi, allinea) necessarie per dislocare opportunamente le varie finestre aperte nella cornice MDI, oppure per passare da una finestra aperta ad un'altra momentaneamente coperta. Il flag WindowList associa alla voce di menu un sottomenu che mostra l'elenco delle finestre/form aperte al momento. Il metodo per provocare l'allineamento o la sovrapposizione delle finestre (che debbono essere aperte) è Arrange, che dispone di quattro varianti.





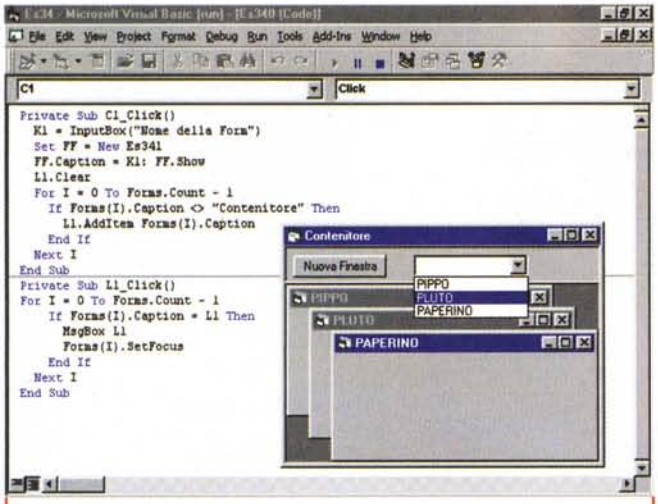


Figura 9 - Visual Basic 5.0 - Generazione automatica di finestre MDIChild. Nell'esercizio precedente i tre form MDIChild erano precostruiti, in questo esercizio vediamo invece come sia possibile generare automaticamente tutti i form MDIChild necessari all'applicazione partendo da un form prototipo. Vediamo anche, ma lo descriviamo nel testo, come sia possibile inserire degli oggetti, nel nostro caso una lista da cui scegliere un elemento e conseguentemente generare un nuovo form, direttamente nel form MDI. In pratica un form MDIChild è una classe (prototipo) dalla quale si generano le varie istanze (copie).

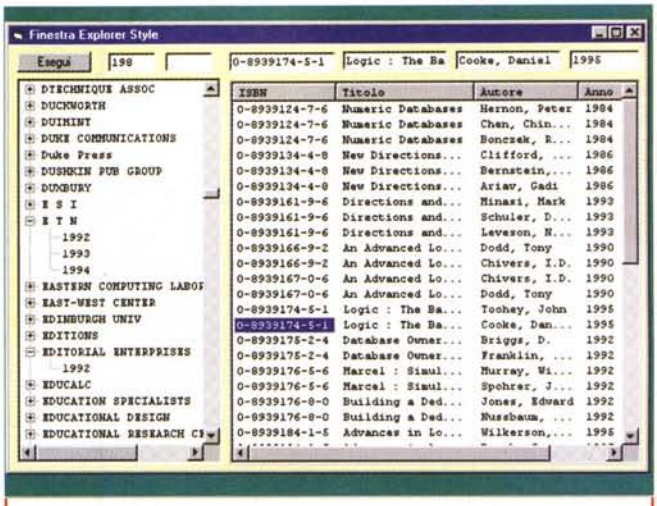


Figura 10 - Visual Basic 5.0 - Riempimento di un form Explorer Style. Questo esercizio è un po' più complesso degli altri per cui siamo costretti a presentarne a parte il listato. Si tratta di un tipico form Explorer Style con a sinistra un controllo TreeView nel quale abbiamo inserito l'elenco degli editori e per ciascuno di essi l'elenco degli anni in cui risulta che abbiano pubblicato dei volumi. Al clic sull'editore nel controllo ListView appaiono i volumi pubblicati (in realtà tutte le accoppiate tra autori e titoli). Se si fa clic sull'anno appaiono i volumi pubblicati in quell'anno.

' in cui Es341 è il nome del form
FF.Show
' il form viene visualizzato
FF.Caption="Titolo"
' e se ne possono modificare le proprietà

Ma l'esercizio mostra altre cose. Ad esempio come sia possibile inserire un controllo di tipo Picture all'interno di un form MDI e come questo possa ospitare altri controlli. Nel nostro caso abbiamo inserito un pulsante "Nuova Finestra" che ci chiede il nome della nuova finestra che stiamo generando e una ListBox che mostra tutti i nomi delle finestre fino ad allora create. La ListBox serve anche per attivare il form selezionato. Notare anche come sia possibile sfruttare la "collection" di form che ogni applicazione mette a disposizione dell'utente.

Qualche regola sui form MDI

Ricapitoliamo alcuni dei concetti relativi ad un'applicazione MDI:

In un'applicazione di tipo MDI esiste un solo form MDI e possono esistere infiniti form Child.

Il form MDI non può ospitare controlli se non di tipo Picture. Se questo viene inserito si posiziona automaticamente in alto nel form MDI. Il controllo di tipo Picture può, a sua volta, ospitare controlli normali.

Per aprire e chiudere form Child si possono usare i soliti metodi Show e Hide, legandoli ai soliti eventi, ad esempio agli eventi di mnu o di ToolBar.

In un'applicazione MDI il menu deve sempre prevedere la voce Finestre, che serve a passare da una finestra aperta ad un'altra, oppure a disporre ordinatamente le stesse.

Normalmente, in un'applicazione MDI, il menu dell'applicazione risiede nel form MDI e non nei vari form Child. Se un form Child dispone di un suo menu, questo viene "ereditato" dal form MDI contenitore quando il form Child viene attivato.

Nel caso di applicazioni ospitanti OLE Containers è possibile "negoziare" la posizione del menu, che può essere trasferito nel form esterno oppure apparire all'interno del contenitore.

Applicazioni Explorer Style

Le applicazioni SDI e MDI prevedono l'uso di più form. Un'applicazione Explorer Style consiste sostanzialmente in un unico form diviso in due parti, quella a sinistra contenente un controllo TreeView (vista ad albero) e quella a destra contenente un controllo ListView. C'è un rapporto di dipendenza tra i due controlli in quanto il contenuto di quello di destra viene dinamicamente aggiornato sulla base di quanto accade nel controllo di sinistra.

Il termine Explorer Style deriva dal gestore delle risorse di Windows (in quello in lingua originale si chiama Explorer) che ben si presta a tantissimi utilizzi applicativi, quelli in cui sia importante anche l'organizzazione dei dati che vengono visualizzati, o la navigazione alla ricerca del dato o del documento giusto.

Se realizzate l'applicazione Explorer Style con la procedura di autocomposizione ottenete un'applicazione già pronta, in cui, però, i due controlli Tree e List non sono "popolati" di dati.

E' utile comunque partire dal form autocomposto in quanto permette di spostare dinamicamente il separatore tra Tree e List, operazione resa possibile dall'utilizzo di un oggetto di tipo Image trasparente.

I due controlli TreeView e ListView fanno parte del gruppo di controlli aggiuntivi Microsoft Windows Common Controls 5.0.

Se non sapete caricare di dati una TreeView o una ListView... ve lo insegniamo noi con i due esercizi di figura 12 e 13, in cui il caricamento avviene "a mano", nel senso che vediamo direttamente nel listato i dati caricati, che non provengono quindi da un database.

```
Dim DD As Database
Dim RR, RS As Recordset
Dim ND As Node
Dim IT As ListItem

Private Sub Form_Load()
  LL.ColumnHeaders.Add , , "ISBN"
  LL.ColumnHeaders.Add , , "Titolo"
  LL.ColumnHeaders.Add , , "Autore"
  LL.ColumnHeaders.Add , , "Anno"
End Sub

Private Sub C1_Click()
On Error Resume Next
Set DD = OpenDatabase("d:\vvvv\terzo\biblio.mdb")
S = "select distinct Publishers.PubID,Publishers.Name,Titles.[Year Published] "
S = S & " From Publishers,Titles "
S = S & " Where Publishers.PubID = Titles.PubID "
S = S & " Order By Publishers.Name,Titles.[Year Published] "
Set RR = DD.OpenRecordset(S, 4)
Do While Not RR.EOF
 K0 = "K" & RR.Fields(0): K1 = RR.Fields(1)
 Set ND = TV.Nodes.Add(, , K0, K1)
  Do While K1 = RR.Fields(1)
   K2 = RR.Fields(2): K3 = K0 & "/" & K2
   Set ND = TV.Nodes.Add(K0, tvwChild, K3, K2)
   RR.MoveNext: If RR.EOF Then Exit Do
  Loop
 Loop
End Sub

Private Sub TV_NodeClick(ByVal Node As Node)
LL.ListItems.Clear
X0 = Node.Key: X1 = InStr(X0, "/")
If X1 > 0 Then
 X2 = Mid(X0, 2, X1 - 2): X3 = Mid(X0, X1 + 1)
Else
 X2 = Mid(X0, 2): X3 = ""
End If
T1 = X2: T2 = X3
S = "Select Titles.ISBN,Titles.Title,Authors.Author,Titles.[Year Published]"
S = S & " from Publishers,Titles,[Title Author],Authors"
S = S & " where Publishers.PubID = Titles.PubID"
S = S & " and Titles.ISBN = [Title Author].ISBN"
S = S & " and [Title Author].Au_ID = Authors.Au_ID"
S = S & " and Publishers.PubID=102"
If X3 <> "" Then S = S & " and Titles.[Year Published]=" & X3
S = S & " Order by Titles.ISBN"
Set RR = DD.OpenRecordset(S, 4)
Do While Not RR.EOF
  Set IT = LL.ListItems.Add
  IT.Text = RR.Fields(0)
  IT.SubItems(1) = RR.Fields(1)
  IT.SubItems(2) = RR.Fields(2)
  IT.SubItems(3) = RR.Fields(3)
 RR.MoveNext
Loop
End Sub

Private Sub LL_ItemClick(ByVal CL As ComctlLib.ListItem)
 T3 = CL.Text
 T4 = CL.SubItems(1)
 T5 = CL.SubItems(2)
 T6 = CL.SubItems(3)
End Sub
```

Figura 11 - Visual Basic 5.0 - Riempimento di un form Explorer Style – Listato.
La complessità del listato dipende dal fatto che vi troviamo sia tutte le istruzioni necessarie a gestire i due controlli TreeView e ListView, che sono abbastanza complicati, sia le istruzioni DAO, squisitamente database, che servono per riempire le due liste con dati letti dal database. In entrambi i casi si utilizzano query, la prima alimentata dalla tabella Publishers e Titles, che produce l'elenco degli editori e degli anni di pubblicazione, e la seconda che mostra dati da tutte e quattro le tabelle del database Biblio. Insomma occorre conoscere i meccanismi delle query.

Sulla TreeView possiamo fare due tipi di click, quello sull'Icona, che serve ad esplodere/implodere il ramo, e quello sull'item (nodo) che serve per selezionare l'item stesso.

L'evento principale è il NodeClick, e le proprietà della TreeView danno tutte le informazione sul nodo cliccato.

In figura 13 vediamo come si carica la ListView. Anche in questo caso conviene dire due o tre cose.

La ListView permette le classiche quattro viste (Icone Grandi, Icone Piccole, Lista e Dettagli). Poiché nelle prime tre si vede solo una informazione alfanumerica, è chiara l'importanza del fatto che ad ogni elemento corrisponde un item (dato principale) e poi vari subitems (dati secondari).

La proprietà con la quale si imposta il tipo di vista è View.

Per associare ad ogni item una icona occorre usare un controllo ImageList, che va riempito con le immagini necessarie (che vengono numerate) e poi associato al controllo ListView. Quando si caricano gli item nella ListView occorre indicare quale icona, quindi quale numero, è ad esso associata. Lo vediamo chiaramente in fi-

Sulla TreeView diciamo due o tre cose.

L'organizzazione dell'albero dipende dalle chiavi.

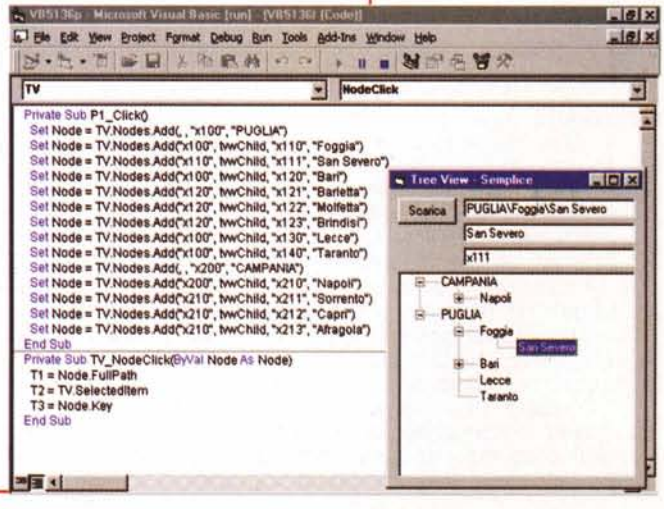
Non è necessario che i dati caricati siano caricati in ordine. L'ordine viene dato dalle chiavi.

La chiave deve essere alfanumerica.

E' possibile caricare e scaricare i dati dinamicamente, al verificarsi di certi eventi. Questo consente (non lo vediamo) di caricare prima il primo livello dell'albero e poi esplodere via via i rami su cui si fa click, oppure scaricare i rami che vengono chiusi.

Per creare i rapporti gerarchici tra gli item dell'albero non è necessario che un elemento terminale abbia la chiave. La chiave stessa può essere, anzi quasi sempre lo è, utile dal punto di vista applicativo. Se ad esempio mostriamo il cognome del dipendente di una azienda si può usare come chiave la sua matricola.

Figura 12 - Visual Basic 5.0 - Come si alimenta un controllo TreeView.
Mostriamo due programmi, più semplici dei precedenti, che hanno la finalità di mostrare come si alimenta un controllo TreeView (nell'esempio si chiama TV). Da notare come occorra utilizzare delle chiavi per indicare i rapporti gerarchici tra un elemento ed i suoi dipendenti. Non è necessario che gli elementi siano caricati nello stesso ordine in cui appaiano, in quanto prevale l'organizzazione imposta dalle chiavi. Non è necessario caricare tutti gli elementi subito ma il caricamento può essere realizzato dinamicamente. Non è indispensabile per il funzionamento dell'albero che gli elementi "terminali" abbiano una chiave. Altre caratteristiche del TreeView le citiamo nel testo.



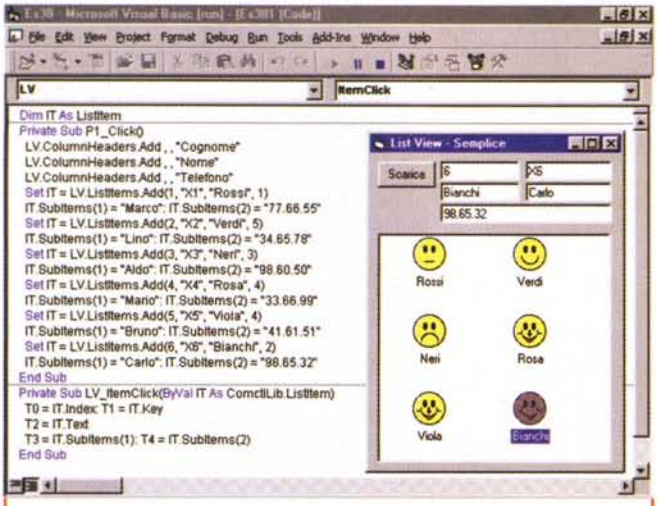


Figura 13 - Visual Basic 5.0 - Come si alimenta un controllo ListView. La compagna del controllo TreeView è la ListView, quella che nel gestore delle risorse di Windows 95 permette le classiche quattro viste (icone grandi, icone piccole, lista e dettagli). Qui ne vediamo una in modalità icone grandi. Nel listato vediamo come si caricano gli elementi della lista, che hanno un index, una key e che possono avere più subitem. Nel testo descriviamo anche come si fa ad associare a ciascun elemento una icona.

Figura 14 - Visual Basic 5.0 - Primo esempio di Explorer Style: il WinHelp di Windows 98. La modalità di visualizzazione dei dati "alla Explorer" è diffusissima. Qui ne vediamo un esempio applicativo chiarissimo, realizzato con il generatore di help di Windows 98, che prevede una organizzazione Explorer Style e che lavora anche con file in formato HTML. Parliamo del generatore di help di Windows 98 in un altro articolo di questo stesso numero di MC.

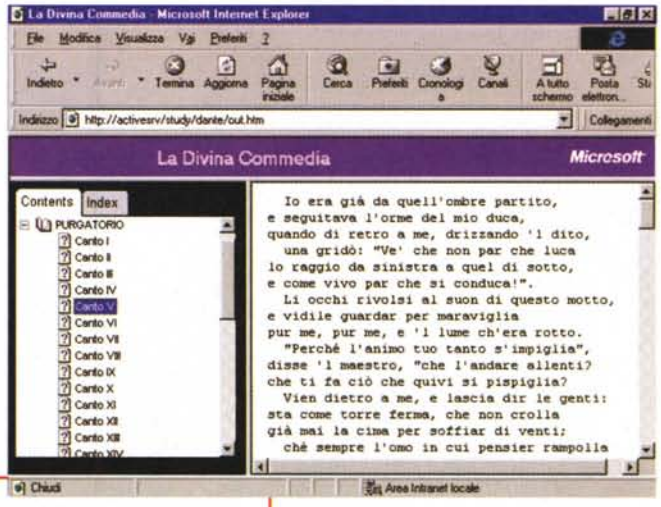


Figure 15,16 - Visual Basic 5.0 - Secondo esempio di Explorer Style - Per navigare in Internet. La struttura ad albero, tipica del controllo TreeView, è ora facilmente praticabile con il DHTML, il nuovo formato delle pagine per Internet. In alternativa si possono usare dei componenti ActiveX o Applet Java, che mostrano oggetti TreeView, alimentabili più semplicemente di quanto non si debba fare con il DHTML. Nel testo parliamo più a fondo di questo argomento. Un fatto è certo: qualsiasi sia lo strumento che usiamo per sviluppare, ad esempio Visual Basic, ad esempio pagine HTML, più o meno potenziate, dobbiamo sempre prevedere come modalità di navigazione sui nostri dati la modalità Explorer Style.

gura 13.

Quando si fa click su un elemento della ListView (evento ItemClick), è possibile leggere la stessa serie di proprietà, indipendentemente dal tipo di vista su cui la ListView stessa è settata.

In conclusione si tratta di due oggetti, la TreeView e la ListView, più complessi degli altri, per quello che fanno. Sono necessariamente un po' più difficili da programmare.

In figura 11 vediamo l'applicazione di quanto detto sui due oggetti ad un caso più complesso, in cui a sinistra si selezionano dei dati che fanno da filtro per i dati che si vedono sulla destra. I dati provengono dal nostro database Biblio.mdb, ed in pratica servono per selezionare dei titoli per editore e per anno di stampa.

Guardiamoci intorno

Per concludere l'articolo mostriamo due figure prese qua e là in altre applicazioni in cui si può usare una modalità di visualizzazione di dati e di documenti Explorer Style. L'help di Windows 98, una pagina DHTML e, sempre in una situazione Internet/intranet, una pagina realizzata con ActiveX, che permettono viste ad albero e permettono dei link tra le due porzioni della videata.

MC

Tutte le autostrade informatiche portano a Roma.

M&CM

IL MODO MIGLIORE PER CONOSCERE IL FUTURO, E VIVERLO.

Dal 4 all'8 dicembre 1998 il futuro vi riserva un appuntamento imperdibile: Webshow! 98. Cinque giorni dedicati a Internet, per comunicare e interagire nella rete, scoprire tutte le novità, i migliori prodotti e servizi dell'Information e Communication Technology.

UN NUOVO MODO DI COMUNICARE.

Tutti i visitatori potranno vivere l'esperienza dell'Internet Cafè, trovare la più ampia scelta di prodotti hardware e software, chiedere il consiglio degli esperti ed entrare in contatto con i più rivoluzionari mezzi di comunicazione, educazione e divertimento.

FUTURO PRESENTE.

Qui trovate il futuro. Webshow! 98 è una grande vetrina attraverso la quale farsi conoscere da tutti, anticipare le novità tecnologiche delle telecomunicazioni e chiarire al grande pubblico i nomi, i ruoli e i prodotti di riferimento nell'ambito delle piattaforme, del software e delle tecnologie di rete.

Webshow! La Fiera dedicata all'Information & Communication Technology.

Roma 4-8 dicembre 1998.

Per informazioni: Roma • Tel. 06/39734421 • e-mail: webshow@fieradiroma.it
Milano • Tel. 02/66034243 • e-mail: webshow@jackson.it

webshow!

Informatica, Networking, Telecomunicazioni.

Organizzazione

Fiera di Roma

GRUPPO EDITORIALE JACKSON