

Corso Intermedio - Lezione 6

http://www.vbsimple.net/intermed/int_06.htm

- [Inizializzazione delle proprietà.](#)
- [L'istruzione With ... End With.](#)
- [L'istruzione Put per la scrittura di dati su file.](#)
- [La funzione LOF.](#)
- [Istruzioni e Funzioni basate sul numero di file.](#)

Nella [lezione 4](#) abbiamo deciso di non preoccuparci troppo delle proprietà dei controlli ed assegnar loro in [fase di progettazione](#) solo quelle proprietà di sola lettura durante la [fase di esecuzione](#). Questo, abbiamo detto, favorisce la [generalizzazione del codice](#) e semplifica il riutilizzo del codice dopo una serie di modifiche.

All'avvio del progetto sarà quindi necessario inizializzare quelle proprietà non regolate durante la progettazione. In particolare, la funzione **Len** consente di determinare la lunghezza di una stringa (soprattutto quando questa è a lunghezza fissa) e tale valore dovrà essere assegnato alla proprietà **MaxLength** di alcune caselle di testo, in modo da limitare l'immissione di testo in ognuna di esse.

L'inizializzazione delle proprietà dei controlli dovrebbe avvenire prima di ogni utilizzo dei controlli, in modo da avere la sicurezza che le operazioni su di essi vengano svolte nella maniera corretta. Non è allora assurdo dire che l'inizializzazione di tutte le proprietà debba avere la priorità massima durante il caricamento del loro form contenitore. Per questa ragione l'operazione sarà effettuata all'interno dell'[evento](#) ⚡ *Load* del Form, come segue:

```

1. Private Sub Form_Load()
2.     txtTitoloBranco.MaxLength = Len(udtDiscoCorrente.strTitoloBranco)
3.     txtAutoreBranco.MaxLength = Len(udtDiscoCorrente.strAutore)
4.     txtTitoloCD.MaxLength = Len(udtDiscoCorrente.strTitoloCD)
5.     txtNote.MaxLength = Len(udtDiscoCorrente.strNote)
6.     txtNumeroTraccia.MaxLength = 2
7.     txtAnnoPubblicazione.MaxLength = 4
8.     cmdEliminaBranco.Enabled = False
9.     cmdModificaBranco.Enabled = False
10.     ....

```

Questo metodo sebbene funzioni e sia anche formalmente corretto non è molto efficiente. Infatti ogni volta che viene fatto riferimento ad un membro di un oggetto o di una struttura con una sintassi così esplicita il compilatore spreca del tempo inutile a valutare il codice alla sinistra del punto che precede il nome del membro: nel nostro esempio si tratta di **udtDiscoCorrente**, che verrà valutato ben 4 volte inutilmente, solo per indicare a quale struttura o oggetto il membro (*strTitoloBranco*, *strAutore*, *strTitoloCD* e *strNote*) appartiene; questa operazione è detta **qualificazione degli identificatori**.

Il linguaggio ci viene incontro con un'istruzione utile a qualificare una volta sola una struttura o un oggetto ed indicare di volta in volta i suoi membri omettendone il nome del

qualificatore.

Ovvero l'istruzione ***With ... End With*** consente indicare l'inizio e la fine di un blocco di codice per il quale viene applicato il concetto di autoqualificazione espresso precedentemente. Il suo funzionamento è estremamente semplice:

```
With <qualificatore>
    .<identificatore> = Oggetto.Height
    If .<identificatore> = valore Then valore = .<identificatore>
End With
```

Naturalmente <qualificatore> indicherà il nome di un oggetto o di una struttura, mentre <identificatore> indicherà un membro del qualificatore.

La prima delle quattro righe definisce il qualificatore da utilizzare per tutti gli identificatori all'interno del blocco, per i quali non è specificato un altro qualificatore. Questo significa che all'interno del blocco ***With ... End With*** potremo comunque utilizzare altri qualificatori per fare riferimento ad altri oggetti o strutture.

La seconda riga infatti assegna il valore della proprietà Height di Oggetto (ovvero di un altro qualificatore) all'identificatore autoqualificato. L'identificatore utilizzato in questa maniera non è soggetto ad alcuna limitazione; la terza riga ne mostra la sua lettura e la sua scrittura, come se il riferimento fosse pienamente qualificato con <qualificatore>.<identificatore>.

Il qualificatore sarà quindi valutato soltanto una volta e tutti i successivi riferimenti abbreviati assumeranno quel qualificatore. Il programma ne guadagnerà in prestazioni quando faremo riferimento ad identificatori autoqualificati almeno due volte all'interno del blocco. È però necessario tenere a mente la rigidità di questa regola: **il qualificatore sarà valutato soltanto una volta all'inizio del blocco.**

Ciò comporta un rischio quando il codice all'interno del blocco autoqualificato tenta di modificare lo stesso qualificatore. Il resto del codice continuerà a fare riferimento al primo qualificatore, perché valutato solo una volta. Ad esempio il codice seguente potrebbe trarre in inganno:

```
1. Dim txtCasella As TextBox
2. Set txtCasella = Text1
3. With txtCasella
4.     MsgBox .Name
5.     Set txtCasella = Text2
6.     MsgBox txtCasella.Name
7.     MsgBox .Name
8. End With
9.
10. With txtCasella
11.     MsgBox .Name
12. End With
```

Alla riga 3 la variabile oggetto ***txtCasella*** indicherà che l'identificatore **Name** utilizzato nelle righe successive appartiene all'oggetto ***Text1***. La riga 5 che dovrebbe assegnare alla variabile ***txtCasella*** l'oggetto ***Text2*** produce il suo effetto, come mostrato dall'esecuzione della riga 6, ma il riferimento autoqualificato della riga 7 continuerà a puntare all'oggetto d'origine (***Text1***).

Il qualificatore sarà nuovamente valutato all'uscita del blocco ***With ... End With***, come

mostrato alle righe 10-12 che mostreranno il valore *Text2*.

Tornando al progetto, il codice di inizializzazione delle proprietà dei controlli potrebbe essere il seguente:

```

1. Private Sub Form_Load()
2.     With udtDiscoCorrente
3.         txtTitoloBranco.MaxLength = Len(.strTitoloBranco)
4.         txtAutoreBranco.MaxLength = Len(.strAutore)
5.         txtTitoloCD.MaxLength = Len(.strTitoloCD)
6.         txtNote.MaxLength = Len(.strNote)
7.     End With
8.     txtNumeroTraccia.MaxLength = 2
9.     txtAnnoPubblicazione.MaxLength = 4
10.    cmdEliminaBranco.Enabled = False
11.    cmdModificaBranco.Enabled = False
12.    ...

```

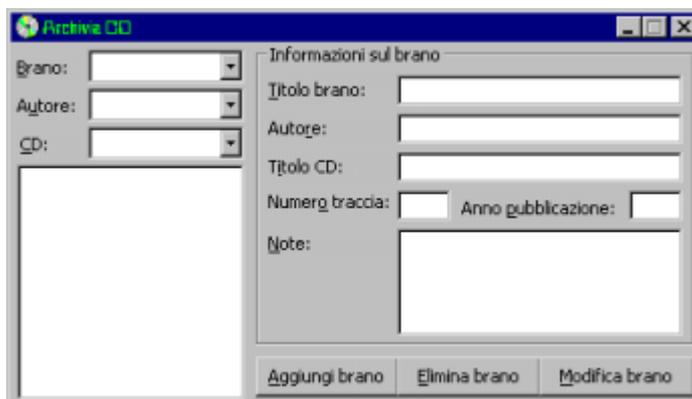
Avremo due righe di codice in più (righe 2 e 7) ma i 4 identificatori all'interno di questo blocco saranno qualificati soltanto una volta, a beneficio della velocità di esecuzione.

Le righe 8-11 non necessitano del qualificatore **udtDiscoCorrente** e pertanto sono state escluse dal blocco *With ... End With*. Avremmo anche potuto inserirle all'interno di quel blocco senza migliorarne né peggiorarne le prestazioni.

Fino a questo punto il nostro progetto non fa nulla di realmente utile.

La prima operazione che svilupperemo sarà l'inserimento di un brano nell'archivio aperto al caricamento del form.

Il codice che svilupperemo sarà quindi legato all'evento  *Click* del controllo **cmdAggiungi**.



L'operazione in sostanza dovrà riempire il record **udtDiscoCorrente** con i valori delle varie caselle di testo e successivamente scrivere l'intero record all'interno del file di dati. Ciò garantirà la corretta scrittura dei dati nella posizione, ordine ed ampiezza corrette.

Tuttavia, almeno all'inizio, non sarà effettuato alcun controllo sui dati inseriti, ovvero sui campi *Numero traccia* ed *Anno pubblicazione*; ci limiteremo a convertirli in numeri e scriverli all'interno del corrispondente campo del record.

La scrittura di dati in un file con accesso Random è effettuata mediante l'istruzione **Put** che utilizza la seguente sintassi:

```
Put [#]<numerofile>, [<posizione>], <dati>
```

Del simbolo # ne discuteremo [subito dopo](#). L'istruzione **Put** richiede naturalmente un **numero di file** già aperto e pronto a ricevere dati, cioè non aperto in modalità *Input*. Il

secondo argomento dell'istruzione consente di specificare la posizione (espressa in numero di record per i files ad accesso Random oppure in numero di bytes per i files ad accesso Binary) in cui andranno scritti i dati. Se la posizione non viene specificata i dati saranno scritti nella posizione di scrittura corrente.

L'ultimo argomento dell'istruzione indica i dati da scrivere e può includere costanti o variabili di tipi elementari o definiti dall'utente, individuali o poste in matrici statiche o dinamiche. Per ognuno di questi tipi di dati saranno applicate delle regole di memorizzazione particolari, includendo un descrittore prima dei dati stessi, ma in questa fase iniziale non è fondamentale conoscere come saranno scritti i dati all'interno del file poiché il tutto avverrà in maniera invisibile al programmatore.

L'istruzione **Put** non consente di scrivere variabili Oggetto in un file.

Non dovremmo avere problemi nell'assegnare un valore al primo ed al terzo argomento dell'istruzione; abbiamo già il numero del file (**intFileDati**) ed i dati da scrivere corrispondono al nostro record (**udtDiscoCorrente**). Qualche problema potrebbe crearcelo l'argomento **<posizione>** che specifica il numero del record in cui andranno scritti i nostri dati.

Un aiuto concreto può fornircelo la funzione **LOF** (*Length Of File*) che come dice il nome, restituisce la lunghezza in bytes del file specificato. La sua sintassi, molto semplice, è:

```
LOF(<numerofile>)
```

Restituisce un valore *Long* che corrisponde al numero di bytes contenuti nel file indicato da **<numerofile>**.

Mediante questa funzione è molto semplice sapere il numero di record a dimensione fissa già presenti all'interno del file. Ad esempio il nostro record a dimensione fissa occupa esattamente 343 bytes, valore ottenibile mediante la funzione **Len** già vista nelle lezioni precedenti. Pertanto un file di 1372 bytes conterrà 4 records (1372 / 343) al suo interno.

Tornando all'istruzione Put, la posizione di scrittura, pertanto, potrà essere calcolata con estrema semplicità facendo riferimento al numero di records già presenti all'interno del file +1.

Alla luce di tutto quanto visto in questa lezione, il codice per l'evento *Click* del pulsante **cmdAggiungi** dovrebbe essere qualcosa del genere:

```
25. Private Sub cmdAggiungiBrano_Click()
26.     With udtDiscoCorrente
27.         .intIndice = LOF(intFileDati) / Len(udtDiscoCorrente) + 1
28.         .strTitoloBrano = txtTitoloBrano.Text
29.         .strAutore = txtAutoreBrano.Text
30.         .strTitoloCD = txtTitoloCD.Text
31.         .intTraccia = Int(txtNumeroTraccia.Text)
32.         .intAnnoPubblicazione = Int(txtAnnoPubblicazione.Text)
33.         .strNote = txtNote.Text
34.         .blnEliminato = False
35.         Put intFileDati, .intIndice, udtDiscoCorrente
36.     End With
37. End Sub
```

Poiché ci saranno parecchi riferimenti alla variabile struttura **udtDiscoCorrente** è più

efficiente e più pratico applicare sui suoi membri l'autoqualificazione mediante **With ... End With**. I suoi membri saranno riempiti con i valori presenti all'interno delle varie caselle di testo, ad eccezione di due soli membri: **intIndice**, che assumerà l'indice del record da scrivere nel file, nella maniera vista precedentemente ($LOF(intFileDati) / Len(udtDiscoCorrente) + 1$) e **blnEliminato** che ovviamente assumerà il valore False per i record appena inseriti.

Una volta riempita la variabile **udtDiscoCorrente** sarà opportuno scriverla sul file mediante l'istruzione Put. Il secondo argomento dell'istruzione (<posizione>) l'abbiamo già ricavato alla riga 27.

Prima di concludere rimane solo un breve punto da sciogliere: le istruzioni e le funzioni basate sul numero di file, quali Open, Close, Put e LOF, utilizzano un sistema un po' contorto; riprendiamo ad esempio la sintassi dell'istruzione Put:

```
Put [#]<numerofile>, [<posizione>], <dati>
```

Quel simbolo # infatti è opzionale e la sua presenza non determina alcuna differenza di comportamento. La stessa regola vale anche per altre **istruzioni** che utilizzano i numeri di file (Open, Close, Put, Get e Seek): il simbolo # può essere omissso tranquillamente.

In maniera opposta, non tutte le **funzioni** che utilizzano i numeri di files gradiscono la presenza del simbolo # ed in alcuni casi generano un errore. Sembrerebbe quindi comportamento normale ignorare la presenza del simbolo # ed ometterlo sia nelle istruzioni che nelle funzioni.

Purtroppo non è così semplice: esistono ancora 3 singole **istruzioni** che si comportano in maniera irregolare. Si tratta delle istruzioni **Print**, **Write** ed **Input**, che per accedere al numero di file specificato pretendono che esso sia preceduto dal simbolo #, in netto contrasto con tutte le altre istruzioni di accesso ai files. Per tale ragione le tre suddette istruzioni sono solitamente chiamate **Print #**, **Write #** ed **Input #**, facendo risaltare il simbolo # che deve precedere il numero di file.

Non sorgano quindi dubbi quando una funzione o un'istruzione del genere usa far precedere dal simbolo # il numero del file cui si riferisce. In conclusione, si tratta di un comportamento decisamente sgradevole... 🚫

Fino a questo punto il progetto sviluppato, al suo avvio inizializza le proprietà dei controlli ed apre il file BRANI.DAT; alla pressione del pulsante **cmdAggiungi** salva sul file tutto ciò che trova nelle varie caselle di testo, senza effettuare alcuna verifica; alla chiusura del form è chiuso anche il file precedentemente aperto.

Vedremo con la prossima lezione come rileggere i dati salvati su file e gestirne il funzionamento collegato ai controlli.

Fibia FBI

26 Gennaio 2003

 [Torna alla quinta lezione](#)

[Vai alla settima lezione](#) 