

# **Visual Basic per Excel**

## **GUIDA COMPLETA v.1**

**Come utilizzare gli oggetti della  
libreria Excel per creare report  
in Microsoft Excel**



## Indice

Preparativi per l'utilizzo della libreria MS Excel.....	3
Inizializzare gli oggetti.....	3
Le celle, il Range e la visibilità di sessione foglio di calcolo e cartella di lavoro.....	5
Lavorare con l'oggetto Range.....	8
Esercizio sull'oggetto Range.....	10
Le operazioni di Taglia, Copia e Incolla sull'oggetto Range.....	11
Esercizio sulle operazioni di Taglia, Copia e Incolla sull'oggetto Range.....	14
La proprietà RangeSelection.....	16
Esercizio sulla proprietà RangeSelection.....	18
Aggiungere un commento ad una cella.....	19
La formattazione delle celle.....	19
Le dimensioni delle celle.....	21
Esercizio sui valori delle celle.....	22
Assegnare una formula alle celle.....	24
Esercizio sulle formule delle celle.....	26
La protezione del foglio di calcolo.....	27
Esercizio sulla protezione del foglio di calcolo.....	28
Importazione di dati esterni.....	29
Esercizio sull'importazione di dati esterni.....	32
Le Visualizzazioni.....	33
Operazioni tra celle.....	34
La curva di Bézier.....	35
Esercizio sulla curva di Bézier.....	36
Importazione di un foglio di calcolo Excel tramite un contenitore OLE.....	37
Riempire le celle con un solo metodo.....	40
Esercizio sui metodi FillDown, FillUp, FillRight e FillLeft.....	42
L'oggetto Scenario e l'insieme Scenarios.....	43
L'oggetto Area e l'insieme Areas.....	48
Riferimenti.....	50

## Preparativi per l'utilizzo della libreria MS Excel

Per poter creare un collegamento tra un'applicazione sviluppata in Visual Basic e Microsoft Excel, è anzitutto necessario consultare la libreria oggetti di Microsoft Excel disponibile nella finestra di dialogo Riferimenti di Visual Basic. Per poter poi aggiungere il riferimento bisogna scegliere Riferimenti dal menu Progetto e porre

premendo il tasto F2 oppure scegliendo Visualizza / Visualizzatore oggetti si può dare un'occhiata preliminare agli oggetti che Excel ha a disposizione e che saranno proprio quelli che si andrà a manipolare.

## Inizializzare gli oggetti

La differenza fondamentale tra gli oggetti Excel e gli oggetti Word visti nel corso di Visual Basic per Word, sta nel fatto che non solo dovremo gestire una sessione di lavoro ed un documento (o meglio una cartella di lavoro), ma anche uno o più fogli per ogni documento.

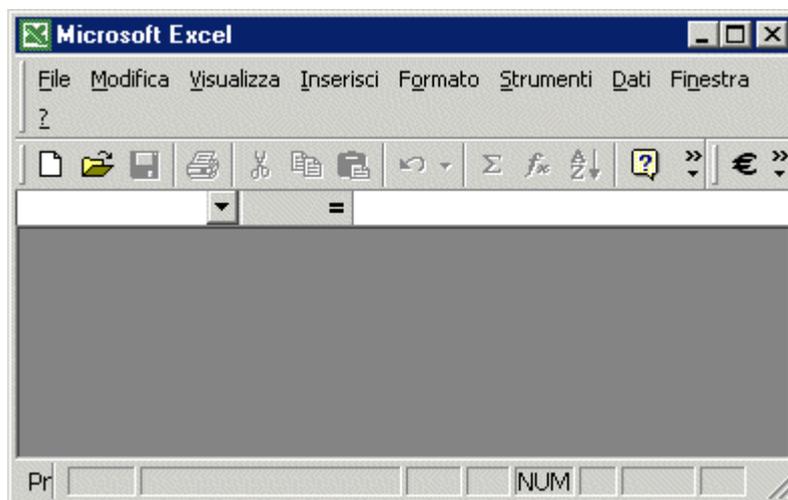
Questo perché, com'è ben noto, ogni cartella di lavoro di Excel è come un fascicolo contenente al suo interno tutti i fogli che lo compongono.

Il grafico in fondo al capitolo dovrebbe spiegare in modo più diretto quale relazione intercorre tra gli oggetti Excel: l'**Applicazione Excel** non è altro che la sessione di lavoro. Si apre una nuova sessione di lavoro avviando Excel. Ci possono dunque essere più sessioni di lavoro aperte contemporaneamente. Come si vedrà poi, dare il nome ad ogni sessione è necessario per riferirsi ad una sessione di lavoro in particolare.

E' importante rendersi conto che in realtà quando normalmente si apre Excel, non solo viene attivata una sessione di lavoro, ma viene creato anche una cartella di lavoro vuoto (che è il secondo oggetto Excel) ed vari fogli di calcolo (che rappresentano il terzo oggetto Excel). Di cartelle di lavoro e fogli di calcolo si parlerà in seguito.

Se volessimo considerare una sessione "pura", il vero oggetto **Application** di Excel, dovremo quindi aprire Excel e dal menu File selezionare la voce Chiudi.

Ecco come si presenta una sessione di Excel al netto degli oggetti Cartella di lavoro e Foglio:



Entriamo un po' più nello specifico: il discorso è astratto ma facilmente riconducibile ad un piano più concreto: si pensi al mondo dell'automobile. Per poter descrivere ad un amico l'auto che abbiamo appena acquistato sarà necessario avere in mente il concetto generale di automobile, ossia l'insieme di tutte le automobili (Oggetto). Una volta definito in generale cos'è un'automobile si può descriverne una in particolare: la marca, il modello, il colore, la cilindrata e così via (Istanza dell'oggetto).

Allo stesso modo si considera l'insieme di tutte le infinite applicazioni Excel (Excel.Application) e se ne sceglie una.

Denominiamo tale applicazione appExcel:

```
Dim appExcel As New Excel.Application
```

Siccome non è prevista l'intercettazione di sessioni di lavoro già esistenti, è stata introdotta nella linea di codice precedente la parola New che indica appunto che l'applicazione appExcel è di nuova creazione.

Adesso che è stato chiarito il concetto di Applicazione, si può passare a definire il termine cartella di lavoro. Un oggetto Cartella di lavoro è l'insieme di tanti fogli di calcolo. Come già visto poco fa, quando solitamente si apre Excel fuori da Visual Basic, viene generato automaticamente una cartella di lavoro vuoto composto da tre fogli anch'essi vuoti.

Aprire Excel da codice invece non prevede tale automatizzazione in quanto a chi sviluppa un'applicazione in Visual Basic è consentito l'accesso alla libreria completa degli oggetti di Excel e quindi la gestione separata di Applicazione, Cartella e Foglio. Per questa ragione sarà dunque necessario, dopo aver aperto una nuova sessione, generare un nuovo foglio oppure aprirne uno esistente sul disco fisso o su qualsiasi altro supporto. Prima di tutto però dovremo prelevare dall'insieme di tutte le cartelle possibili (Oggetto) ossia Excel.Workbook la specifica cartella che c'interessa definire (Istanza dell'oggetto).

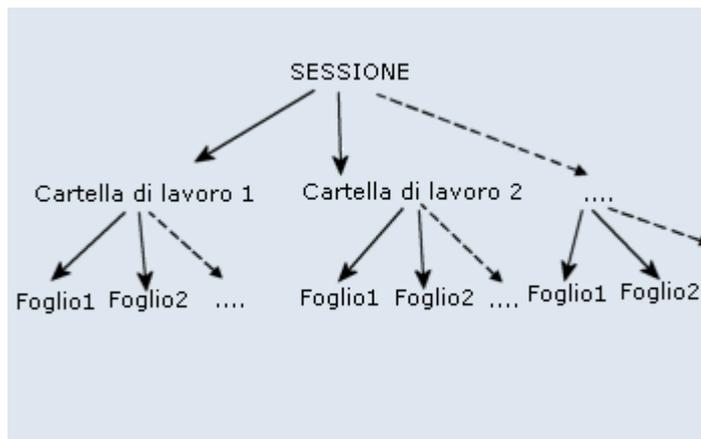
Chiameremo tale oggetto cartExcel:

```
Dim cartExcel As Excel.Workbook
```

Perchè non abbiamo inserito la parola New come nel caso di appExcel? Perchè non necessariamente una cartella dev'essere di nuova creazione: si può ad esempio aprire una cartella Excel archiviata nel disco fisso ma non è possibile fare altrettanto con una sessione di lavoro Excel. Questo per il semplice motivo che quando si salva il lavoro non viene salvata la sessione ma la cartella ed i fogli ad essa associati.

Si può adesso passare alla terza definizione, quella cioè di Foglio. Un foglio di lavoro (o foglio di calcolo) è costituito dall'insieme delle celle sulle quali si può lavorare. L'elenco dei fogli di lavoro è visualizzabile attraverso le schede dei fogli.

Ecco finalmente il grafico esplicativo:



Definiamo tra tutti i fogli di calcolo possibili (Oggetto) uno specifico foglio (Istanza dell'oggetto) denominato foglioExcel:

```
Dim foglioExcel As Excel.Worksheet
```

Ora che tutte le istanze sono state definite, si può procedere all'apertura della sessione e della cartella di lavoro. La creazione di una nuova cartella di lavoro, essendo nient'altro che una serie di fogli di calcolo, apre automaticamente i fogli stessi.

D'ora in poi non si farà più distinzione tra oggetto ed istanza dell'oggetto. Siccome la definizione di appExcel esprime già la volontà di aprire una nuova sessione di lavoro, non sarà necessario aprire una nuova applicazione Excel.

Sarà sufficiente visualizzarla inserendo nella routine Load di Form1 la seguente linea di codice:

```
Private Sub Form_Load()  
appExcel.Visible = True  
End Sub
```

Dando il via all'applicazione appena scritta si aprirà una sessione di lavoro vuota da ogni cartella proprio come mostrava la prima immagine del capitolo.

Adesso andiamo incontro alla prima scelta: aprire una cartella di lavoro esistente oppure crearne una vuota? Nel primo caso sarà necessario indicare il percorso del file .xls presente sul disco fisso e successivamente rendere la cartella visibile:

```
Set cartExcel = Excel.Workbooks.Open("C:\Documenti\cartella.xls")
```

nel secondo caso sarà sufficiente indicare quanto segue:

```
Set cartExcel = Excel.Workbooks.Add
```

Aprire un nuovo foglio di lavoro è un'operazione simile a quella appena vista:

```
Set foglioExcel = Excel.Worksheets.Add
```

Notare che un nuovo foglio (foglio4) si è aggiunto ai tre fogli aperti per default alla creazione dell'oggetto Cartella di lavoro.

Per aprire un foglio di calcolo esistente la procedura è notevolmente diversa. Innanzitutto si deve aprire o creare la cartella di lavoro nella quale selezionare il foglio ed in seguito scegliere il foglio tra quelli disponibili:

```
Set cartExcel = Excel.Workbooks.Add  
Set foglioExcel = Excel.Worksheets.Item(2)
```

In questo caso abbiamo scelto di aprire il secondo foglio di calcolo di una nuova cartella. Per concludere questo capitolo aggiungiamo che è possibile rinominare il foglio selezionato attraverso la proprietà Name associata agli oggetti Excel.Worksheets.

Rinominiamo ad esempio il primo foglio di una nuova cartella di lavoro come "Primo foglio":

```
Set cartExcel = Excel.Workbooks.Add  
Set foglioExcel = Excel.Worksheets.Item(2) foglioExcel.Name = "Primo foglio"
```

## Le celle, il Range e la visibilità di sessione foglio di calcolo e cartella di lavoro

Le linee di codice analizzate nel capitolo precedente possono risultare piuttosto scomode nel momento in cui si va ad avviare l'applicazione per testarla, visto che alla chiusura del programma, la sessione di Excel (e naturalmente la cartella di lavoro ed il foglio di calcolo) rimarranno aperti, obbligando ogni volta ad una chiusura manuale.

E' possibile quindi associare all'evento di chiusura dell'applicazione Visual Basic la chiusura della sessione Excel e della cartella di lavoro.

Come si vedrà nel corso del capitolo sorgerà però un problema: la chiusura di una cartella di lavoro è sempre accompagnata dalla richiesta di salvataggio della stessa.

Così, anche se la sessione di lavoro Excel è invisibile, cioè se è stato impostato:

```
appExcel.Visible = False
```

comparirà una finestra di messaggio al di fuori dell'applicazione che abbiamo eseguito che chiede se si desidera salvare il documento prima di chiuderlo.

Evitare quest'intrusione è molto semplice: basta indicare da codice se si vuole salvare la cartella di lavoro alla chiusura oppure no.

Nel primo caso dev'essere naturalmente indicato un percorso valido su disco fisso o qualsiasi altro supporto. Cominciamo però aggiungendo che è possibile gestire la visibilità anche dei singoli fogli di lavoro. Per permettere la visualizzazione di un singolo foglio di lavoro (ad esempio Foglio2, ossia il secondo foglio di lavoro dei tre creati per default alla creazione di una nuova cartella di lavoro) è sufficiente inserire le seguenti linee di codice:

```
Set foglioExcel = Excel.Worksheets.Item(2)  
foglioExcel.Visible = xlSheetVisible
```

Con la prima linea si indica a quale foglio di lavoro dei disponibili applicare lo stato di visualizzazione, definito invece dalla seconda riga.

Per impedire la vista ad esempio del primo foglio di lavoro:

```
Set foglioExcel = Excel.Worksheets.Item(1)  
foglioExcel.Visible = xlSheetHidden
```

Importante è anche il metodo **Activate** che indica se un oggetto è attivo oppure no. E' importante dal momento che si può indicare come è stato fatto sopra in quale foglio di calcolo lavorare ma questo non comporta visualizzazione automatica di tale foglio.

Ad esempio se decidessimo di rendere visibile l'applicazione Excel e di lavorare su Foglio 2, sarebbe opportuno passare alla visualizzazione di Foglio2. Ecco come fare:

```
Set foglioExcel = Excel.Worksheets.Item(2)
foglioExcel.Activate
```

Adesso si può passare finalmente alla chiusura del documento. Quello che dovremo fare è chiudere la cartella di lavoro nella quale sono contenuti tutti i fogli di calcolo.

Per eseguire tale operazione sarà sufficiente un semplice:

```
Private Sub Form_Unload(Cancel As Integer)
cartExcel.Close False
End Sub
```

Notare che non appare nella seconda linea di codice il segno di uguale (=) prima di False perché False è in questo caso un parametro che si assegna all'operazione di chiusura.

Avendo indicato False come parametro, abbiamo indicato che alla chiusura della cartella di lavoro non si desidera che appaia la richiesta di salvataggio della stessa.

Il salvataggio di una cartella di lavoro infatti è un'operazione a parte:

```
cartExcel.Save
```

salva la cartella corrente in base al nome assegnatole. Se quindi abbiamo aperto una cartella di lavoro esistente, questa sarà aggiornata con i nuovi dati dell'ultima sessione di lavoro. In caso contrario, essendo il nome di default Cartel1, Cartel2...Carteln, la cartella di lavoro sarà salvata con tale nome.

Diverso è il caso della seguente linea di codice:

```
cartExcel.SaveAs "C:\Documenti\Cartella.xls"
```

nella quale viene espressamente indicato il nome del percorso da seguire per il salvataggio della cartella di lavoro.

Una volta chiusa la cartella di lavoro rimarrà però aperta la sessione Excel, proprio come mostrato dalla prima figura del capitolo precedente.

Sarà necessario allora aggiungere:

```
appExcel.Quit
```

Adesso passiamo alla gestione delle celle. E' possibile considerare una cella alla volta indicandone le coordinate ossia il numero di riga ed il numero di colonna. Ad esempio la prima cella in alto a sinistra sarà indicata come:

```
foglioExcel.Cells(1, 1)
```

Da sola questa linea di codice non ha nessun significato. Serve però a definire la posizione della cella desiderata.

Ancora più importante è capire la proprietà-oggetto **Range**. Essa infatti definisce un'area di validità ossia in poche parole il numero di celle che compongono l'area che desideriamo attivare o comunque considerare.

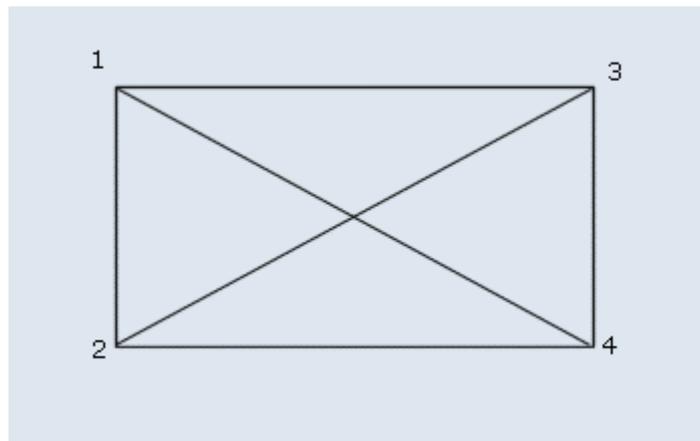
Se decidiamo infatti di cancellare il contenuto delle prime due caselle in alto a sinistra quindi rispettivamente le celle di coordinate (1,1) e (1,2), dovremo considerare un Range ossia un'area di validità che ha inizio dalla prima casella e termina nella seconda.

Probabilmente la seguente figura chiarirà meglio il significato della proprietà: consideriamo l'esempio fatto poc'anzi di un Range di celle (1,1) e (1,2):

	A	B	C	D
1	RANGE			
2				
3				
4				
5				
6				

Come si sarà compreso, il Range è rappresentabile graficamente da un rettangolo. E proprio come un rettangolo è possibile definirne le dimensioni solamente tramite una delle due coppie punti opposti dalla diagonale scelta.

Ad esempio, per definire il rettangolo del Range potremo scegliere i punti 1 e 4 oppure i punti 2 e 3 come mostra la figura sotto:



Proviamo prima di chiudere questo capitolo a selezionare sul foglio di lavoro il Range composto dalle celle (2,1) e (5,3):

```
foglioExcel.Range(foglioExcel.Cells(2, 1), foglioExcel.Cells(5, 3)).Select
```

eccone il risultato:

	A	B	C	D
1				
2				
3				
4				
5				
6				
7				

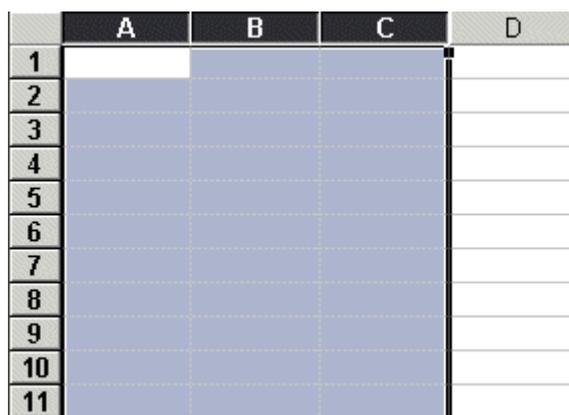
## Lavorare con l'oggetto Range

Una volta definita l'area di validità o Range del foglio di calcolo, si può pensare ad interagire con essa ad esempio selezionando tutte le colonne del foglio stesso che ospitano almeno una delle celle del Range. Per completare tale operazione sarà sufficiente sostituire al metodo **Select** di Range il metodo **Select** collegato alla proprietà **EntireColumn**, come mostrato dal codice qui sotto:

```
foglioExcel.Range(foglioExcel.Cells(2, 1), foglioExcel.Cells(5, _  
3)).EntireColumn.Select
```

dove è stato scelto di selezionare tutte le celle delle colonne che contengono parte del Range di dimensioni (2,1) - (5,3).

Il risultato, da un punto di vista puramente grafico è il seguente:



	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				

Adesso sarà necessario chiarire due punti importanti. Il primo, relativo all'immagine sopra è molto banale: per un motivo di spazio l'area selezionata va ben oltre l'undicesima riga, in quanto l'immagine è stata tagliata per evidenti motivi di spazio. Teoricamente il metodo **EntireColumn.Select** ed il corrispondente metodo da applicare alle righe (che vedremo subito dopo), permettono di selezionare tutte le celle delle colonne, fino alla fine del foglio di lavoro.

Altro discorso è quello dell'area di validità. Essa infatti è rappresentata da Excel dal rettangolo con bordo in grassetto ed area blu, come s'è già visto in una figura del capitolo precedente.

Selezionando il set di colonne del Range (2,1) - (5,3) si è deciso di ampliare l'area di validità passando dal Range (2,1) - (5,3) al Range (1,1) - (+oo, 3). Ancora meglio: parlando in termini molto più consoni al linguaggio di sviluppo che utilizziamo, la nuova area di validità è rappresentata dall'area (1,1) - (foglioExcel.Columns.Count, 3) dove:

```
foglioExcel.Columns.Count
```

rappresenta il conto totale delle colonne presenti nel foglio di calcolo. Esattamente allo stesso modo è possibile tenere il conto delle righe esistenti:

```
foglioExcel.Rows.Count
```

Così come sono state selezionate tutte le colonne, è altresì possibile attivare tutte le righe che contengono almeno una delle celle che compongono l'area di Range assegnata:

```
foglioExcel.Range(foglioExcel.Cells(2, 1), foglioExcel.Cells(5, _  
3)).EntireRow.Select
```

Anche qui il risultato dell'operazione è facilmente intuibile:

	A	B	C	D	E
1					
2					
3					
4					
5					
6					

Si noti che anche in questo specifico caso valgono le indicazioni relative all'esempio mostrato prima ossia: l'area di validità o di selezione va ben oltre le cinque colonne indicate dalla figura in quanto il Range sarà rappresentato da un rettangolo di dimensioni (2,1) (5,+oo)

Vediamo adesso come inserire i dati in ognuna delle celle del foglio di lavoro: dovrà essere utilizzata la proprietà **Text** di Characters.

Questo significa che sarà necessario indicare la cella come combinazione di riga e colonna e quindi scrivere:

```
foglioExcel.Cells(1, 1).Characters().Text = "Prima cella"
```

Il codice qui sopra aggiunge il testo "Prima cella" alla cella di coordinate (1, 1). E' necessario ricordare che il primo di ogni coppia di valori che determinano le celle indica la riga ed il secondo la colonna.

Come già è stato visto nel capitolo precedente, è opportuno per semplificare e velocizzare l'apprendimento degli esempi proposti, avere sempre sott'occhio il foglio di lavoro sul quale si sta lavorando.

Così, per nostra convenienza, possiamo decidere di lavorare sul primo foglio di calcolo e conseguentemente attivarlo. Sarà quindi più semplice notare che "Prima cella" è ora il testo della cella (1,1):

```
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
foglioExcel.Cells(1, 1).Characters().Text = "Prima cella"
```

Stesso risultato di prima si ottiene indicando i valori di riga e colonna delle celle non in modo espresso ma attraverso variabili.

In questo modo sarà sicuramente più facile gestire il codice modificando tali variabili, senza dover andare ad operare su istruzioni lunghe ed articolate. Se vogliamo ad esempio riprodurre il risultato ottenuto in precedenza, ossia la selezione di un Range di celle (2,1) - (5,3):

```
Dim Colonna1, Colonna2, Riga1, Riga2 As Integer
'si può facilmente operare su questa linea di codice
'modificando i valori di ciascuna variabile
Riga1 = 2: Colonna1 = 1: Riga2 = 5: Colonna2 = 3
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2, _
Colonna2)).Select
```

In questo modo sarà oltretutto più semplice andare a popolare l'area di selezione, ossia ad inserire valori (numerici, testuali) in ciascuna delle celle che compongono il Range.

Proviamo con un metodo progressivo ossia mediante l'utilizzo di un ciclo For...Next. Il ragionamento è il seguente: per il valore di un contatore che chiamiamo i che va dalla riga della prima cella in alto a sinistra del Range alla riga della cella in basso a sinistra del Range facciamo variare un secondo contatore, n, che dalla prima colonna all'ultima del Range popola tutte le celle della riga:

```
For i = Riga1 To Riga2
For n = Colonna1 To Colonna2
foglioExcel.Cells(i, n).Characters().Text = "Cella " & i & " " & n
Next n
Next i
```

Nota: essendo il codice mostrato sopra del tutto indipendente dal Range, non è detto che non si possa popolare una serie di celle differente dal Range stesso.

Ad esempio nulla vieta di popolare tutte le celle della prima colonna, selezionando invece tutte le celle della prima riga.

Questo è davvero un punto importante per le lezioni che verranno, nelle quali si discuterà come selezionare un'area non vuota (nella quale le celle non abbiano valore "") e come copiare, incollare, tagliare, raggruppare, calcolare i valori delle celle che si sono selezionate.

	<b>Esercizio</b>
---	------------------

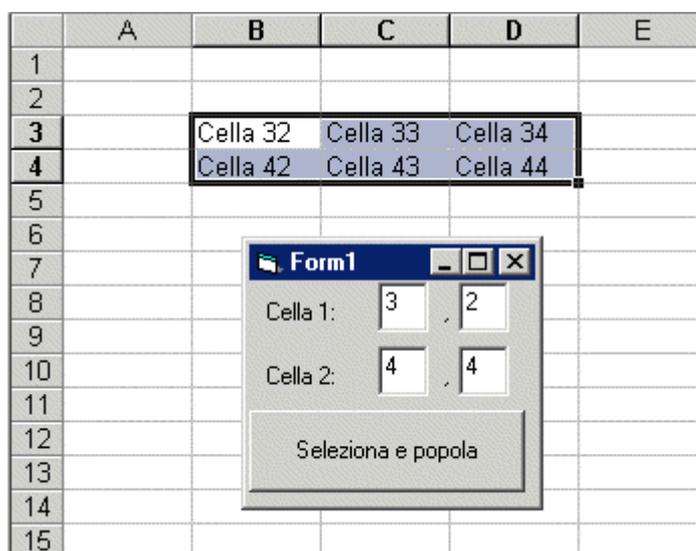
Sviluppare una semplice applicazione che:

1. visualizzi applicazione Excel, cartella di lavoro e che operi e visualizzi il secondo foglio di lavoro;
2. chieda all'utente le coordinate del Range di selezione (nota: come visto nel capitolo precedente il Range si determina con due sole celle opposte da una delle due diagonali del rettangolo di selezione) e selezioni tale Range; Nota importante: se si utilizzano le caselle di testo per permettere all'utente di inserire le coordinate delle celle, si ricordi che il loro contenuto è in forma testuale perciò non compatibile con i valori numerici delle coordinate. Per rendere il contenuto di una TextBox compatibile, indicare ad esempio:

```
Riga1 = Val(TextBox1.Text)
```

3. popoli tutto il Range selezionato con il testo "Cella coordinata riga, coordinata colonna come visto nel corso del capitolo.

Ecco un esempio di come può essere impostata l'applicazione:



	<b>Soluzione</b>
---	------------------

```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet

Private Sub Command1_Click()
Dim Colonna1, Colonna2, Riga1, Riga2 As Integer
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(3)
foglioExcel.Activate
Riga1 = Val(Text1.Text): Colonna1 = Val(Text2.Text)
Riga2 = Val(Text3.Text): Colonna2 = Val(Text4.Text)
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2,
Colonna2)).Select
For i = Riga1 To Riga2
For n = Colonna1 To Colonna2
foglioExcel.Cells(i, n).Characters().Text = "Cella " & i & n
Next n
```

```
Next i
End Sub
```

## Le operazioni di Taglia, Copia e Incolla sull'oggetto Range

Riprendendo le conclusioni alle quali si era arrivati nel corso del capitolo precedente, è possibile vedere ora come agire sul Range attraverso le operazioni tipiche di Taglia, Copia ed Incolla.

La struttura di queste operazioni è simile per tutte e tre e molto comprensibile: sarà infatti sufficiente utilizzare i rispettivi metodi.

Vediamone il primo, ossia il metodo **Cut**. Cut permette di tagliare una selezione ossia tutte le caselle che fanno parte di un Range.

La struttura da utilizzare segue il modello indicato qui sotto:

```
OggettoExcel.Range(cella1, cella2).Cut destinazione
```

Non è tuttavia assolutamente necessario operare una selezione di un Range e poi richiamare il metodo Cut, sebbene nulla vieti di compiere entrambi i passaggi.

Una cosa infatti è la selezione di un Range, un'altra è l'operazione di taglio delle caselle che lo compongono. Così, selezionando e tagliando un Range si ottiene il risultato mostrato dalla figura seguente:

	A	B	C	D	E	F
1	Cella 1					
2		Cella 2				
3			Cella 3			
4				Cella 4		
5					Cella 5	
6						
7						
8						

considerando il fatto che l'immagine è stata prodotta da un codice nel quale non è stata indicata alcuna destinazione, ossia:

```
Riga1 = 1: Colonna1 = 1: Riga2 = 5: Colonna2 = 5
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2,
Colonna2)).Select
For i = Riga1 To Riga2
foglioExcel.Cells(i, i).Characters().Text = "Cella " & i & " " & n
Next i
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2,
Colonna2)).Cut
```

Eliminando invece la selezione del Range (ossia la seconda linea del codice mostrato sopra) si otterrà un risultato di questo tipo:

	A	B	C	D	E	F
1	Cella 1					
2		Cella 2				
3			Cella 3			
4				Cella 4		
5					Cella 5	
6						
7						
8						

Dove vanno a finire le selezioni tagliate? Nel caso in cui non si indichi la destinazione verranno memorizzate nella Clipboard ossia negli Appunti.

In caso contrario, si dovrà scegliere un Range di pari dimensioni. Ad esempio se la selezione è formata da un rettangolo di 5x5 come nella figura riportata sopra, sarà sufficiente indicare una destinazione di dimensioni 5x5.

Ad esempio, proviamo a traslare le caselle del Range (1,1) - (5,5) di tre caselle più a destra, semplicemente aggiungendo il valore 3 ad ogni riga e colonna come mostrato dal codice sotto:

```
Riga1 = 1: Colonna1 = 1: Riga2 = 5: Colonna2 = 5
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2, Colonna2)).Select
For i = Riga1 To Riga2
foglioExcel.Cells(i, i).Characters().Text = "Cella " & i & " " & n
Next i
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2, Colonna2)).Cut _
foglioExcel.Range(foglioExcel.Cells(Riga1 + 3, Colonna1 + 3), foglioExcel.Cells(Riga2 + 3, Colonna2 + 3))
```

Si otterrà un risultato simile a quello mostrato in figura:

	A	B	C	D	E	F	G	H
1								
2								
3								
4				Cella 1				
5					Cella 2			
6						Cella 3		
7							Cella 4	
8								Cella 5
9								
10								

Tale risultato è possibile perchè è stato indicato un Range di destinazione delle stesse dimensioni di quello di origine.

L'operazione di copia, ottenibile attraverso il metodo Copy e del tutto simile a quella appena vista.

Ed anche la sintassi da utilizzare non cambia di molto:

```
OggettoExcel.Range(cella1, cella2).Copy destinazione
```

Anche in questo caso infatti se *destinazione* non viene indicato, le caselle facenti parte del *Range* vengono copiate negli Appunti.

L'operazione Incolla ottenuta attraverso il metodo **PasteSpecial** incolla il contenuto degli Appunti sul foglio utilizzando un formato specifico, in modo tale da poter utilizzare dati di formato differente in base all'applicazione di provenienza degli stessi.

Non è detto infatti che necessariamente si debba incollare i dati presenti nelle caselle dello stesso foglio di lavoro all'interno del quale si sta lavorando.

La sintassi è del tipo:

```
OggettoExcel.Range(cella1, cella2).PasteSpecial xlPasteType, xlPasteSpecialOperation, Skip Blanks, Traspose
```

dove *xlPasteType* è un argomento facoltativo di tipo *Variant* che indica una delle operazioni disponibili tra quelle elencate nella tabella sottostante:

Tipo di operazione	Risultato ottenuto
xlPasteAll	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il Range di origine se indicato.
xlPasteAllExceptBorders	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il Range di origine se indicato, senza i bordi o le linee eventualmente disegnate tra le caselle nel Range di origine.
xlPasteComments	Incolla tutti i commenti aggiunti alle celle che compongono il Range e solo essi.

	Quindi il contenuto delle celle non viene incollato.
xIPasteFormats	Incolla il formato del Range di origine (ad esempio con rientro, senza rientro, con tabulazioni incrociate eccetera). Anche questa operazione non incolla il contenuto delle celle.
xIPasteValues	Incolla il valore delle celle specificate. Se l'oggetto Range contiene più celle, verrà restituita una matrice di valori.

Se si passa a considerare il gruppo di operazioni indicato come xIPasteSpecialOperation nella struttura vista prima, si può notare che anch'esso è un argomento facoltativo di tipo Variant che specifica l'operazione di incollamento.

Può essere rappresentato da una delle costanti mostrate nella tabella sottostante:

Tipo di operazione	Risultato ottenuto
xIPasteSpecialOperationAdd	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il <i>Range</i> di origine se indicato in un nuovo <i>Range</i> sommando i valori delle caselle già presenti a quelli delle caselle che devono essere incollate.
xIPasteSpecialOperationSubtract	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il <i>Range</i> di origine se indicato, in un nuovo <i>Range</i> sottraendo i valori delle caselle che devono essere incollate da quelli delle caselle già presenti.
xIPasteSpecialOperationMultiply	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il <i>Range</i> di origine se indicato in un nuovo <i>Range</i> moltiplicando i valori delle caselle già presenti per quelli delle caselle che devono essere incollate.
xIPasteSpecialOperationDivide	Incolla tutto il contenuto degli Appunti se i dati sono stati memorizzati in essi oppure tutto il <i>Range</i> di origine se indicato in un nuovo <i>Range</i> dividendo i valori delle caselle già presenti per quelli delle caselle che devono essere incollate.
xIPasteSpecialOperationNone	E' l'impostazione predefinita, l'operazione <i>PasteSpecial</i> corrisponde ad una semplice sostituzione tra i valori esistenti e quelli da incollare.

E' importante notare che se i parametri xIPasteType e xIPasteSpecialOperation devono essere compatibili se si decide di indicare espressamente quest'ultimo.

C'è incompatibilità ad esempio se xIPasteType è costituito dalla costante xIPasteFormats siccome risulterebbe impossibile compiere operazioni matematiche sulla formattazione delle caselle.

Pienamente accettati sono invece xIPasteValues e xIPasteAll. Prima di continuare facciamo un piccolo esempio: copiamo il solito Range (1,1) - (5,5) su se stesso ossia sul Range di destinazione (1,1) - (5,5). Se volessimo ad esempio sommare i valori dovremmo utilizzare la costante xIPasteSpecialOperationAdd, con l'accortezza di sostituire ai valori testuali che si utilizzavano prima ("Cella1", "Cella2"...), dei valori numerici:

```
Riga1 = 1: Colonna1 = 1: Riga2 = 5: Colonna2 = 5
For i = Riga1 To Riga2
foglioExcel.Cells(i, i).Characters().Text = i & n
Next i
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2, Colonna2)).Copy
foglioExcel.Range(foglioExcel.Cells(Riga1, Colonna1), foglioExcel.Cells(Riga2, Colonna2)).PasteSpecial _
xIPasteAll, xIPasteSpecialOperationAdd
```

valori iniziali delle celle erano 1,2,3,4,5. Dopo il copia-incolla con la somma dei valori il risultato è il seguente:

	A	B	C	D	E	F
1	2					
2		4				
3			6			
4				8		
5					10	
6						
7						
8						

mentre una moltiplicazione (quindi l'utilizzo della costante `xlPasteSpecialOperationMultiply` produrrebbe il seguente risultato:

	A	B	C	D	E	F
1	1					
2		4				
3			9			
4				16		
5					25	
6						
7						
8						

Passando oltre, *Skip Blanks* permette, se indicato con valore *True* di non incollare le celle vuote del *Range* di origine nel *Range* di destinazione.

Infine *Transpose*, anch'esso elemento facoltativo. Se è indicato ed ha valore *True*, le righe e le colonne vengono trasposte quando viene incollato l'intervallo.



1) utilizzare il seguente codice per attivare sessione, cartella e foglio e popolare parte del foglio di lavoro

```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet

Private Sub Form_Load()
Dim Colonna1, Colonna2, Riga1, Riga2 As Integer
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Characters().Text = i & n
Next n
Next i
End Sub
```

2) alla pressione del tasto Command1:

- copiare tutte le caselle non vuote ossia il Range (1,1) - (10,10) negli Appunti;
- eseguire l'operazione Incolla Speciale (PasteSpecial) sullo stesso Range di origine (1,1) - (10,10) utilizzando la costante `xlPasteAll` ed un'operazione (`xlPasteSpecialOperationAdd`, `xlPasteSpecialOperationSubtract`, `xlPasteSpecialOperationMultiply`, `xlPasteSpecialOperationDivide`) scelta dall'utente attraverso una ComboBox.

Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F	G	H	I
1	22	24	26	28	30	32	34	36	38
2	42	44	46				54	56	58
3	62	64	66				74	76	78
4	82	84	86				94	96	98
5	102	104	106				114	116	118
6	122	124	126				134	136	138
7	142	144	146				154	156	158
8	162	164	166				174	176	178
9	182	184	186				194	196	198
10	202	204	206				214	216	218
11									
12									
13									

Form1

Operazione da compiere:

xlPasteSpecialOperationAdd

Incolla speciale



```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
With Combol
.Text = "xlPasteSpecialOperationAdd"
.AddItem "xlPasteSpecialOperationAdd"
.AddItem "xlPasteSpecialOperationSubtract"
.AddItem "xlPasteSpecialOperationMultiply"
.AddItem "xlPasteSpecialOperationDivide"
End With
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Characters().Text = i & n
Next n
Next i
End Sub
Private Sub Command1_Click()
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(10, 10)).Copy
Select Case Combol.Text
Case Is = "xlPasteSpecialOperationAdd"
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(10, 10)).PasteSpecial xlPasteAll, _
xlPasteSpecialOperationAdd
Case Is = "xlPasteSpecialOperationSubtract"
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(10, 10)).PasteSpecial xlPasteAll, _
xlPasteSpecialOperationSubtract
Case Is = "xlPasteSpecialOperationMultiply"
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(10, 10)).PasteSpecial xlPasteAll, _
xlPasteSpecialOperationMultiply
Case Is = "xlPasteSpecialOperationDivide"
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(10, 10)).PasteSpecial xlPasteAll, _
xlPasteSpecialOperationDivide

```

```
Case Else
End Select
End Sub
```

## La proprietà RangeSelection

In questo capitolo si andrà ad analizzare la proprietà **RangeSelection** come area selezionata dall'utente. Come si vedrà in seguito esiste una differenza sostanziale tra la gestione di un Range, controllato dallo sviluppatore e di un RangeSelection le cui dimensioni e posizione sono invece affidate alle scelte dell'utente. Prima di iniziare con tale argomento, essendo stati già chiariti i fondamenti del concetto di cella e di Range è possibile semplificare ulteriormente le cose utilizzando il metodo utilizzato da Excel per l'individuazione delle celle.

Si può infatti indicare una cella indicandone le coordinate in base all'intestazione di riga e di colonna.

Così la cella che finora chiamavamo (3,3) per indicare la cella nella terza riga e nella terza colonna può essere indicata come "C3".

C infatti corrisponde all'intestazione della terza colonna e 3 all'intestazione della terza riga. Proprio come si vede dall'immagine che segue:

	A	B	C	D	E	F
1						
2						
3			C3			
4						
5						
6						
7						
8						

Quindi, riportando quanto appena visto al concetto di Range è possibile semplificarne la definizione indicando anche qui per ogni cella l'intestazione di riga e di colonna.

Ad esempio se volessimo selezionare l'area (1,1) - (4,3) invece della scrittura finora utilizzata ossia:

```
foglioExcel.Range(foglioExcel.Cells(1, 1), foglioExcel.Cells(4, 3)).Select
```

potremmo utilizzare come indicatori di cella rispettivamente "A1" e "D3" nel modo che segue:

```
foglioExcel.Range("A1", "D3").Select
```

rendendo così molto più agevole la scrittura e la comprensione del codice.

Terminata questa parentesi vediamo adesso come recuperare le informazioni relative ad un'area di selezione, introducendo in questo modo il concetto di RangeSelection. Se volessimo infatti visualizzare in una finestra di messaggio la formula che indica il Range selezionato, dovremo fare riferimento alla proprietà **ActiveWindow** dell'oggetto Application.

ActiveWindow ha la funzione di spostare l'attenzione sulla finestra attiva dell'applicazione Excel, nel caso ne siano state aperte più di una.

Nel caso in cui nessuna finestra dell'applicazione sia aperta, la proprietà ritornerà il valore Nothing.

Un sistema per evitare errori di questo tipo consiste nel verificare immediatamente che almeno una finestra sia aperta.

Ecco come fare:

```
If appExcel.ActiveWindow Is Nothing Then
Msgbox "Nessuna finestra è attiva"
End If
```

Insieme a tale proprietà dovremo fare uso anche della RangeSelection che restituisce un oggetto Range rappresentante le celle selezionate nel foglio di lavoro nella finestra specificata (nel nostro caso la finestra attiva).

Una volta compreso il significato di tali proprietà aggiungiamo che per raggiungere il risultato prefissato, si dovrà utilizzare anche la proprietà Address della quale per adesso non faremo un'analisi dettagliata in

quanto non ancora necessaria.

Unendo le tre proprietà possiamo quindi recuperare la formula che individua il Range attraverso il codice qui sotto:

```
MsgBox appExcel.ActiveWindow.RangeSelection.Address
```

Un esempio di formula visualizzato dalla finestra di messaggio potrebbe essere il seguente:



In questo caso è stato impostato prima un Range di celle "A1","D3" come mostrato dal codice riportato sotto:

```
foglioExcel.Range("a1", "d3").Select  
MsgBox appExcel.ActiveWindow.RangeSelection.Address
```

Naturalmente nel caso in cui non sia stato selezionato espressamente un Range, il risultato mostrato dalla finestra di messaggio corrisponde alla formula di selezione di una sola cella, o la "A1" per impostazione predefinita (formula \$A\$1), oppure quella selezionata dall'utente.

Le proprietà appena viste ed il codice analizzato sopra hanno uno scopo ben specifico. Indicando un Range da codice si suppone che chi sviluppa l'applicazione conosca quali celle andrà a selezionare l'utente, cosa pressoché impossibile.

Attraverso la proprietà **RangeSelection** che per semplificarne il concetto si può considerare come un vero e proprio oggetto indipendente ossia l'oggetto "area selezionata dall'utente" a differenza di Range che può essere considerato più come l'oggetto "area selezionata dallo sviluppatore", si può appunto lavorare con le celle via via selezionate dall'utente dell'applicazione.

Così risulta semplice applicare i concetti visti negli articoli precedenti a tale area di selezione: prendiamo il caso del metodo Taglia. Ecco come si adatta a RangeSelection

```
appExcel.ActiveWindow.RangeSelection.Cut
```

Lo stesso vale per il metodo **Copy**:

```
appExcel.ActiveWindow.RangeSelection.Copy
```

e per Incolla Speciale, di cui per comodità si riporta sotto la sintassi completa:

```
ApplicazioneExcel.ActiveWindow.RangeSelection.PasteSpecial xlPasteType,  
xlPasteSpecialOperation, Skip Blanks, Transpose
```

Se però dal punto di vista della sintassi i tre metodi appena visti a Range e RangeSelection non sono granché dissimili, per l'organizzazione del codice le cose sono molto diverse.

Negli esempi fatti negli articoli precedenti (e negli esercizi finora proposti) non ha senso sostituire semplicemente il codice relativo a RangeSelection al codice relativo a Range perché in questo modo non si lascerebbe all'utente la possibilità di eseguire la selezione.

Il metodo migliore per ovviare al problema è quello far eseguire all'utente la selezione ed applicare poi i metodi Taglia, Copia ed Incolla Speciale nel momento desiderato dall'utente.

Tutto ciò può essere realizzato con l'inserimento nel progetto di un semplice pulsante.

E questo è proprio il testo dell'esercizio proposto al termine del capitolo.



1) utilizzare il seguente codice per attivare sessione, cartella e foglio e popolare parte del foglio di lavoro

```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
Dim Colonna1, Colonna2, Riga1, Riga2 As Integer
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Characters().Text = i & n
Next n
Next i
End Sub

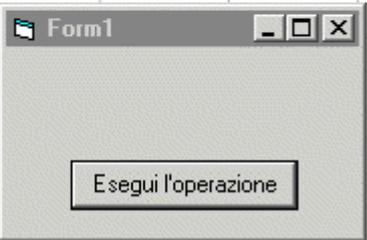
```

2) alla pressione del tasto Command1:

□ attivare e visualizzare il foglio di calcolo 2 e copiarvi la selezione indicata dall'utente partendo dalla cella (1,1). Ricordarsi di utilizzare questa volta *RangeSelection* al posto di *Range* e di copiare la selezione del primo foglio di lavoro negli Appunti prima di cambiare foglio;

Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F
1	36	37	38	39		
2	46	47	48	49		
3	56	57	58	59		
4	66	67	68	69		
5	76	77	78	79		
6	86	87	88	89		
7	96	97	98	99		
8	106	107	108	109		
9						
10						
11						
12						
13						
14						
15						
16						


```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Characters().Text = i & n
Next n
Next i

```

```

Next i
End Sub
Private Sub Command1_Click()
appExcel.ActiveWindow.RangeSelection.Copy
Set foglioExcel = Excel.Worksheets.Item(2)
foglioExcel.Activate
appExcel.ActiveWindow.RangeSelection.PasteSpecial xlPasteValues
End Sub

```

## Aggiungere un commento ad una cella

Altro metodo interessante da considerare è sicuramente **AddComment**. Esso permette infatti di aggiungere un commento ad una cella.

L'aggiunta del commento può essere però eseguita solo singolarmente ossia cella per cella e non selezionando un Range di più celle.

La presenza di un commento relativo ad una cella viene visualizzato nel foglio di calcolo tramite una linguetta rossa all'angolo della cella.

Passando il mouse su tale angolo, appare un ToolTip che visualizza il testo del commento. Quello mostrato in figura è un esempio:

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						

Com'è facile intuire, non necessariamente un commento deve essere incluso in una cella al quale sia stato aggiunto un testo o una formula.

Per aggiungere il commento "Questo è il commento alla cella C2" alla cella C2, appunto:

```
foglioExcel.Cells(2, 3).AddComment "Questo è il commento alla cella C2"
```

se si desidera lavorare sull'oggetto Cells oppure:

```
foglioExcel.Range("C2").AddComment "Questo è il commento alla cella C2"
```

se invece si agisce sul Range unitario di cella C2.

## La formattazione delle celle

E' anche interessante inserire un testo in una cella andando automaticamente a capo quando si raggiunge il limite di questa, come mostrato dalla figura:

	A	B	C	D	E	F
1						
2			Questa cella permette di andare a capo			
3						
4	Questa cella non permette di andare a capo					
5						
6						
7						
8						

Il risultato si ottiene semplicemente impostando la proprietà WrapText su True in questo modo:

```
foglioExcel.Range("C2").WrapText = True
```

Anche in questo caso è possibile attivare la proprietà per un gruppo di celle, indicando ad esempio il Range C2-E4:

```
foglioExcel.Range("c2", "e4").WrapText = True
```

Per impostare un valore da assegnare ad una cella abbiamo visto un metodo in precedenza. Esiste però il metodo ufficiale, che questa volta assegna valori veri e propri e non caratteri alfanumerici come nel caso della proprietà Characters, ossia l'assegnazione della proprietà Value ad un oggetto Range. Così, se si desidera assegnare il valore 123,32 alla cella C2 ed il valore 29 alla cella C4:

```
foglioExcel.Range("C2").Value = 3.14159
foglioExcel.Range("C4").Value = 29
```

Attraverso la proprietà Value è possibile quindi impostare le operazioni sulle celle. Se ad esempio si desiderasse assegnare alla cella D3 il valore corrispondente al prodotto dei valori delle celle C2 e C4:

```
With foglioExcel
.Range("C2").Value = 3.14159
.Range("C4").Value = 29
.Range("D3").Value = .Range("C2").Value * .Range("C4").Value
End With
```

Oppure è possibile prima popolare una serie di celle come già abbiamo fatto negli esercizi precedenti e poi effettuare una ricerca selezionando tutti i valori superiori ad uno dato.

Proviamo a svolgere questo piccolo esercizio passo per passo.

Dopo aver popolato il Range al solito modo (mi ricordandosi della differenza tra Characters e Value):

```
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Value = i * 0.2332
Next n
Next i
```

possiamo quindi copiare nel foglio di calcolo 2 (supposto che prima si stesse lavorando su un foglio di calcolo differente, al fine di evitare confusione inutile), tutti i valori delle celle che superano ad esempio il valore 1.

Quindi, definiamo l'**oggetto Cella** che servirà poi per effettuare la ricerca:

```
Set Cella = appExcel.Cells
```

per ogni Cella nel Range che c'interessa ossia A1 H8:

```
For Each Cella In foglioExcel.Range("A1", "H8")
```

se il valore della singola cella è maggiore di 1:

```
If Cella.Value > 1 Then
```

cambiamo il foglio di calcolo sul quale lavoriamo scegliendo il foglio 2 ed attiviamolo:

```
Set foglioExcel = excel.Worksheets.Item(2)  
foglioExcel.Activate
```

consideriamo ora un semplice contatore che ci servirà per incollare le celle con valore maggiore di 1 tutte in una colonna, in modo progressivo:

```
Cont = Cont + 1
```

e finalmente copiamo le celle:

```
foglioExcel.Cells(Cont, 1).Value = Cella.Value
```

chiusura di cicli e condizioni:

```
End If  
Next Cella
```

Il risultato dovrebbe essere il seguente:

	A	B	C	D	E	F	G
1	1,166						
2	1,166						
3	1,166						
4	1,166						
5	1,166						
6	1,166						
7	1,166						
8	1,166						

dove naturalmente quello visualizzato è Foglio2 e le celle incollate vanno ben oltre l'immagine che è stata tagliata per questioni di spazio.

## Le dimensioni delle celle

Concludiamo con la proprietà **AutoFit** che come si intuisce dal nome, modifica le dimensioni delle colonne e/o delle righe per adattare il contenuto delle celle.

AutoFit è una proprietà associabile soltanto agli oggetti **Columns** e **Rows** perciò in un Range bisogna indicare se applicare la proprietà alla colonna, alla riga o a tutte e due.

Proviamo ad esempio ad inserire un numero molto lungo in una cella ed uno molto corto in un'altra e a ridimensionare le rispettive colonne:

```
foglioExcel.Range("d3").Value = 1  
foglioExcel.Range("b3").Value = 5499030246  
foglioExcel.Range("A1", "d4").Columns.AutoFit
```

Il risultato sarà il seguente:

	A	B	C	D	E	F
1						
2						
3		5499030246		1		
4						
5						
6						
7						
8						



1) utilizzare il seguente codice per attivare sessione, cartella e foglio e popolare parte del foglio di lavoro

```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Value = i & n
Next n
Next i
End Sub
```

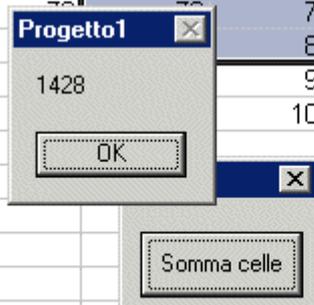
2) alla pressione del tasto Command1:

- visualizzare una finestra di messaggio la somma di tutti i valori delle celle selezionate dall'utente.

Ricordarsi di utilizzare l'oggetto RangeSelection.

Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F	G
1	11	12	13	14	15	16	17
2	21	22	23	24	25	26	27
3	31	32	33	34	35	36	37
4	41	42	43	44	45	46	47
5	51	52	53	54	55	56	57
6	61	62	63	64	65	66	67
7	71	72	73	74	75	76	77
8	81	82	83	84	85	86	87
9	91	92	93	94	95	96	97
10	101	102	103	104	105	106	107
11							
12							
13							
14							
15							
16							





```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
For i = 1 To 10
For n = 1 To 10
foglioExcel.Cells(i, n).Value = i & n
Next n
Next i
End Sub
Private Sub Command1_Click()
Dim Tot As Integer
Set Cella = appExcel.Cells
For Each Cella In ActiveWindow.RangeSelection.Cells
Tot = Tot + Cella
Next Cella
MsgBox Tot
End Sub
```

## Assegnare una formula alle celle

Nel corso di questo capitolo si analizzeranno i metodi di assegnazione delle formule alle celle o ad un Range di celle e l'utilizzo delle funzioni supportate da Excel presenti nell'oggetto **WorksheetFunction**.

Sarà in questo modo possibile accedere attraverso un'applicazione sviluppata in Visual Basic alle funzioni di Excel di tipo matematico e trigonometrico, logico, finanziario, statistico, e così via.

Lasciamo quindi per un attimo da parte il discorso di layout del foglio di lavoro e vediamo una delle più importanti proprietà supportate da Excel: la proprietà **Formula**.

Attraverso tale proprietà s'imposta e si assegna una formula alla cella desiderata in modo da far dipendere il valore della cella (quello che poi sarà visualizzato al suo interno) dalla formula stessa.

Ad esempio consideriamo tre celle. Due di queste, ossia la cella A1 e la cella A2 contengono dei valori numerici che impostiamo a priori.

Consideriamo A1 con valore 10 e A2 con valore 25. Possiamo quindi assegnare il valore alla cella A3 in base ad una formula del tutto arbitraria.

Facciamo un esempio pratico. Assegnamo ad A1 e A2 i rispettivi valori:

```
foglioExcel.Range("A1").Value = 10
foglioExcel.Range("A2").Value = 25
```

adesso assegnamo una formula alla cella A3. Possiamo ad esempio pensare di inserire la formula che esegue l'operazione di somma tra i valori delle celle A1 e A2:

```
foglioExcel.Range("A3").Formula = "=$A$1+$A$2"
```

Anche se non necessaria, la figura sotto mostra il risultato visualizzato:

	A	B	C	D	E	F
1	10					
2	25					
3	35					
4						
5						
6						
7						
8						

dove, selezionando la cella A3, nella casella della formula appare proprio la formula da noi inserita:

A3						
		= =\$A\$1+\$A\$2				
	A	B	C	D	E	F
1	10					
2	25					
3	35					
4						
5						

Allo stesso modo possiamo generare formule utilizzando tutti gli operatori desiderati. Proviamo ora a moltiplicare le celle A1 e A2 e dividerle per la loro differenza:

```
foglioExcel.Range("A3").Formula = "=( $A$1*$A$2) / ($A$2-$A$1) "
```

il risultato sarà:

	A	B	C	D	E	F
1	10					
2	25					
3	16,66667					
4						
5						
6						
7						
8						

Così possiamo calcolare il valore massimo tra quello contenuto nella cella A1 e quello della cella A2:

```
foglioExcel.Range("A3").Formula = "=MAX(A1:A2) "
```

Le funzioni principali di Excel sono tutte contenute nell'oggetto **WorksheetFunction**. La differenza tra l'utilizzo della proprietà Formula e di WorksheetFunction sta nel fatto che mentre il primo inserisce la formula nella casella delle formule di Excel ed esegue l'operazione in modo del tutto automatico, il secondo inserisce il risultato come valore della cella.

Sarebbe del tutto inutile andare ad elencare la lunga lista di funzioni supportate, quindi rimandiamo questo compito ad uno studio della Guida in linea di Microsoft Excel.

Si parlava dell'oggetto WorksheetFunction: il suo utilizzo è davvero semplice, ammesso che si abbia sotto mano una tabella di tutte le funzioni o meglio, dei loro significati.

Soltanto così si ottiene una corrispondenza tra una funzione dell'oggetto ed il risultato desiderato. Senza andare oltre vediamo come ottenere lo stesso risultato dell'esempio precedente utilizzando questo nuovo oggetto:

```
foglioExcel.Range("A3").Value =
appExcel.WorksheetFunction.Sum(foglioExcel.Range("A1"), _
foglioExcel.Range("A2"))
```

E così via per tutte le altre funzioni.

E' interessante anche notare che tra le funzioni contenute in WorksheetFunction esiste la possibilità di generare numeri casuali.

E questo ci renderà le cose più interessanti nella formulazione degli esercizi proposti d'ora in poi. Per generare uno o una serie di numeri casuali si utilizza la funzione **RAND()** di Excel:

```
foglioExcel.Range("A3").Formula = "=RAND()"
```

per prendere soltanto la parte intera di tali numeri casuali:

```
foglioExcel.Range("A3").Formula = "=INT(RAND())"
```

quindi per generare un numero casuale compreso tra 0 e 100:

```
foglioExcel.Range("A3").Formula = "=RAND()*100"
```

ed infine per generare un numero casuale compreso tra due intervalli diversi da 0, ad esempio 20 e 60:

```
foglioExcel.Range("A3").Formula = "=RAND()*(60-20)+20"
```

E' da notare che l'utilizzo di una formula non è ristretta ad una singola cella: un'intero *Range* può essere popolato da celle con la medesima formula.

Ad esempio applichiamo la formula di somma delle celle A1 (che ha sempre il valore 10) e A2 (che ha sempre il valore 25) ad un Range di celle A3,D5

```
foglioExcel.Range("A3", "D5").Formula = "=$A$1+$A$2"
```

Si otterrà il seguente risultato:

	A	B	C	D	E	F
1	10					
2	25					
3	35	35	35	35		
4	35	35	35	35		
5	35	35	35	35		
6						
7						
8						

Allo stesso modo è possibile ad esempio popolare un Range di numeri casuali, passaggio che sarà oggetto dell'esercizio associato a questo capitolo.



- 1) aprire ed attivare una nuova sessione, una cartella ed un foglio di calcolo Excel.
- 2) all'avvio dell'applicazione selezionare il primo foglio di lavoro, renderlo attivo e popolare il Range A2,I9 con numeri casuali interi compresi nell'intervallo (10; 90).
- 3) se il valore di una cella è compreso tra i valori (10; 20) aggiungere un commento alla cella indicando la formula della stessa. Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F	G	H	I
1									
2	14	57	21	79	14	44	16	77	36
3	19	52	12	60	76	10	77	85	15
4	29	16	53	11	41	66	16	24	39
5	70	27	35	29	55	71	55	43	55
6	33	65	33	83	70	53	17	35	44
7	40	17			30	11	58	14	60
8	88	85			20	71	59	66	26
9	83	75			76	81	88	57	33
10									



```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
foglioExcel.Range("A2", "I9").Formula = "=INT(RAND()*(90-10)+10)"
Set cella = appExcel.Cells
For Each cella In foglioExcel.Range("A2", "I9").Cells
If cella.Value > 10 And cella.Value < 20 Then
cella.AddComment (cella.Formula)
End If
Next cella
End Sub

```

## La protezione del foglio di calcolo

Quando il foglio di lavoro viene protetto, è possibile aggiungere una nuova forma di sicurezza che consiste nel rendere invisibili le formule delle celle, anche se queste vengono selezionate dall'utente. Per eseguire tale operazione si utilizza la proprietà **FormulaHidden** che si chiama in causa nel seguente modo:

```
foglioExcel.Range("A1", "D4").FormulaHidden = True
```

questo dimostra che non è necessario procedere cella per cella ma si può agevolmente nascondere la formula di un intero Range di celle. Naturalmente per disabilitare la proprietà basta impostarla su *False*. Se ad esempio selezioniamo un Range di celle A1,D4 e assegniamo ad esso una formula comune ad esempio il prodotto tra il numero di riga e quello di colonna di ogni cella, possiamo renderci conto del risultato ottenuto mediante la proprietà **FormulaHidden**, in questo modo:

```

Set Cella = appExcel.Cells
For Each Cella In foglioExcel.Range("A1", "D4")
Cella.Formula = Cella.Column & Cella.Row
Next Cella
foglioExcel.Range("A1", "D4").FormulaHidden = True

```

Naturalmente per ottenere un qualsiasi risultato il foglio di calcolo deve essere protetto. Quali sono le principali modalità di protezione di un documento Excel? Innanzitutto esiste la proprietà **ProtectWindows** di sola lettura e quindi non modificabile da codice, che determina se le finestre delle cartelle di lavoro sono protette o meno. Poi abbiamo la proprietà **ProtectStructure** anch'essa di sola lettura che determina se la disposizione dei fogli di lavoro è bloccata.

**ProtectSelection** permette di impostare o eliminare la protezione sugli elementi del grafico per cui possono risultare non selezionabili tramite mouse.

**ProtectScenarios** invece indica solo se gli scenari del foglio di lavoro sono protetti.

**ProtectionMode** indica se la protezione a interfaccia utente è attiva.

**ProtectGoalSeek** permette di impostare o eliminare la protezione sulle coordinate del grafico.

**ProtectFormatting** permette di impostare o eliminare la protezione sulla formattazione degli elementi del grafico.

**ProtectDrawingObjects** determina se gli oggetti-disegno sono protetti.

**ProtectData** permette di impostare o eliminare la protezione sull'accesso alle formule delle celle.

**ProtectContents** determina se tutto il contenuto di un foglio è stato protetto o meno.

Nel caso dell'esempio precedente non è possibile utilizzare nessuna di queste proprietà per applicare la protezione al foglio di lavoro siccome sono proprietà di sola lettura.

Si dovrà così ricorrere al metodo **Protect**, dotato della seguente sintassi:

```
Oggetto.Protect(Password, DrawingObjects, Contents, Scenarios, UserInterfaceOnly)
```

dove in *Password* si dovrà indicare la parola chiave di protezione. In *DrawingObjects* si indicherà invece (impostandolo su True o False) se la protezione si estende anche agli elementi di disegno del foglio di calcolo.

*Contents* si riferisce invece all'oggetto in questione (nel nostro caso Oggetto è il foglio di calcolo ossia foglioExcel). Nel caso di un foglio di calcolo questo parametro protegge tutte le celle.

*Scenarios* protegge gli scenari, impostando il parametro su True mentre *UserInterfaceOnly* protegge l'interfaccia utente.

Ad esempio se aggiungessimo all'esempio precedente la seguente linea di codice:

```
foglioExcel.Protect ("password")
```

otterremmo l'invisibilità delle formule di ciascuna cella come mostrato sotto:

	A1		=			
	A	B	Barra della formula	E	F	
1	11	21	31	41		
2	12	22	32	42		
3	13	23	33	43		
4	14	24	34	44		
5						
6						

ed in più, ad ogni tentativo di modificare il valore di qualsiasi cella, otterremmo una finestra di messaggio come quella qui sotto:



Per estendere la protezione anche alla cartella di lavoro sarà sufficiente indicare come oggetto non più foglioExcel ma cartExcel ossia l'oggetto Excel.WorkBook. Una linea di codice di questo tipo:

```
cartExcel.Protect ("password")
```

protegge tutta la cartella di lavoro. Per verificarne il grado di protezione si utilizza la proprietà HasProtection: visualizziamo una finestra di messaggio all'apertura della cartella di lavoro nel caso sia protetta:

```
If cartExcel.HasProtection = True Then MsgBox "La cartella è protetta"
```

Per determinare se la cartella di lavoro è protetta dalla scrittura (anche se tutte le formule possono essere visibili e gli oggetti-disegno o i grafici possono essere disponibili alle azioni del mouse dell'utente) si usa invece la proprietà WriteReserved:

```
If cartExcel.WriteReserved = True Then MsgBox "La cartella è protetta"
```

Nell'esercizio proposto al termine del capitolo si potrà fare un po' di pratica nella protezione di un foglio di lavoro.

Infine vediamo il metodo **PurgeChangeHistoryNow** che permette di eliminare l'elenco delle ultime azioni dal registro delle modifiche della cartella di lavoro.

La sintassi è la seguente:

```
cartExcel.PurgeChangeHistoryNow(Days, SharingPassword)
```

dove *Days* specifica per quanto tempo si desidera mantenere le modifiche in termini di giorni e *SharingPassword* specifica la password, a meno che questa non sia già stata applicata caso in cui verrà ignorata.



- 1) aprire ed attivare una nuova sessione, una cartella ed un foglio di calcolo Excel.
- 2) all'avvio dell'applicazione selezionare il primo foglio di lavoro, renderlo attivo e popolare il Range A1,H8 con numeri casuali interi compresi nell'intervallo (10; 30).
- 3) applicare al foglio di lavoro una password determinata dalla somma di tutti i valori delle celle non vuote (Range A1,H8). 4) proteggere il foglio di lavoro dalla visualizzazione delle formule delle celle Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F	G	H
1	29	14	11	18	27	16	13	11
2	20	25	10	29	13	19	19	26
3	20	13	14	14	14	25	13	19
4	15	17	16	27	22	19	25	26
5	28	29	11	22	23	17	14	27
6	18	10	13	19	19	18	28	16
7	28	11	18	19	19	26	26	13
8	24	19	26	26	11	25	19	22



```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
foglioExcel.Range("A1", "H8").Formula = "=INT(RAND()* (30-10)+10) "
Dim Password As Integer
foglioExcel.Range("A1", "H8").FormulaHidden = True
```

```

Password = appExcel.WorksheetFunction.Sum(foglioExcel.Range("A1"),
foglioExcel.Range("A2"))
foglioExcel.Protect (Password)
End Sub

```

## Importazione di dati esterni

Può risultare particolarmente utile per chi sviluppa una macro o chi utilizza fogli Excel nel proprio progetto andare ad importare un testo, una lista o più in generale una serie di dati da un documento esterno.

Esiste un metodo molto efficace per evitare di compiere complesse operazioni manuali (apertura del file, estrazione di ogni riga, chiusura del file, esame di ogni dato estratto). Il metodo in questione è **OpenText**. Attraverso OpenText viene aperto e analizzato un file di testo come nuova cartella di lavoro con un singolo foglio contenente i dati del file di testo analizzato. Quindi, unica limitazione del metodo, non è possibile importare direttamente il contenuto di un file di testo in un foglio di lavoro esistente.

Ecco come si presenta la sintassi di OpenText:

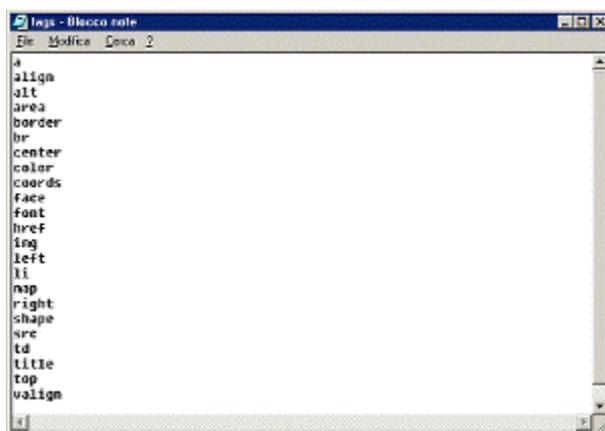
```

appExcel.Workbooks.OpenText(Filename, Origin, StartRow, DataType, TextQualifier,
ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo,
DecimalSeparator, ThousandsSeparator)

```

Vediamo ora di analizzarla con calma: prima di tutto si nota che il metodo è applicabile all'insieme *Workbooks* dell'istanza appExcel.

Il primo parametro obbligatorio, *Filename* è il nome del file di testo da aprire. Ad esempio se creiamo un file tags.txt nella cartella "C:\Documenti" nel quale inseriamo una lista qualsiasi, come la seguente:



dovremo indicare il percorso "C:\Documenti\tags.txt". Il secondo parametro richiesto, *Origin* è invece opzionale ed indica la piattaforma di provenienza del file. Le scelte sono limitate alle costanti del gruppo *xlPlatform* ossia *xlMacintosh*, *xlWindows* oppure *xlMSDOS*.

Il parametro *StartRow* indica invece il numero della riga dalla quale partire nel processo di importazione del testo dal file .txt al foglio di lavoro Excel.

Così se ad esempio si indica *StartRow* = 2, verranno importate tutte le righe a partire dalla seconda.

Anche il parametro *DataType* è opzionale: indica il formato di colonna dei dati. Il formato può essere a scelta *xlDelimited* se si desidera che il file sia delimitato da caratteri delimitatori oppure *xlFixedWidth* se i dati del file devono essere ordinati in colonne di larghezza fissa.

Andando oltre troviamo il parametro *TextQualifier*, anch'esso opzionale che imposta il qualificatore di testo che avvisa l'applicazione che i dati racchiusi da tale qualificatore sono in formato testo. Ad esempio si può impostare il qualificatore di testo come virgoletta singola " " impostando *TextQualifier* su *xlTextQualifierSingleQuote*, come virgoletta doppia " " impostandolo su *xlTextQualifierDoubleQuote* oppure disabilitando questa opzione indicando *xlTextQualifierNone*.

Altro parametro sempre opzionale è *ConsecutiveDelimiter* che, impostato su *True* considera due delimitatori consecutivi (ad esempio due virgolette singole una dietro l'altra " ") come un delimitatore unico.

Naturalmente questo non vale se il parametro *TextQualifier* è stato ignorato oppure se ha valore *xlTextQualifierNone*.

*Tab*, se impostato su *True* considera il carattere di tabulazione come delimitatore.

*Semicolon*, se impostato su *True* considera il carattere punto e virgola " ; " come delimitatore.

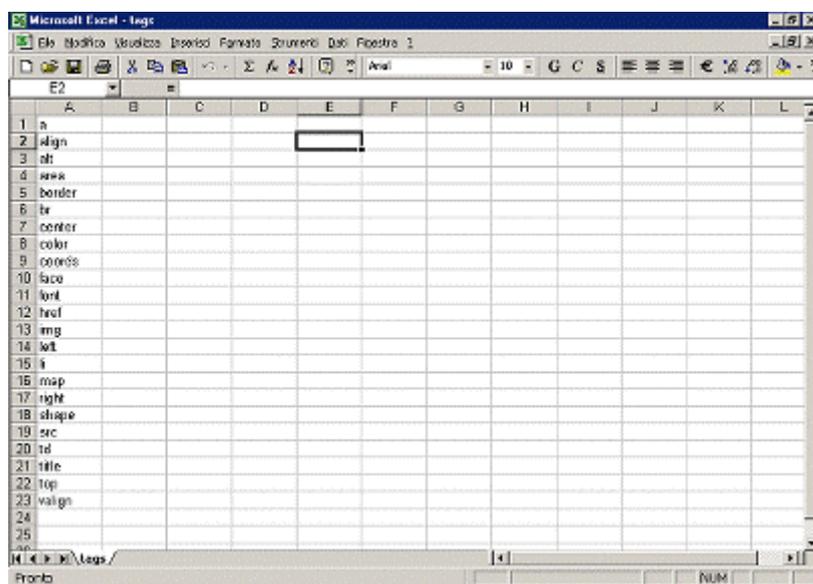
Comma, se impostato su True considera il carattere virgola " , " come delimitatore.  
 Space, se impostato su True considera il carattere spazio " " come delimitatore.  
 Other, se impostato su True consente di utilizzare come delimitatore il carattere indicato dal parametro successivo ossia OtherChar. Se vengono specificati più caratteri, verrà utilizzato solo il primo carattere della stringa e gli altri caratteri verranno ignorati.  
 FieldInfo consente invece di indicare una matrice contenente informazioni di analisi per singole colonne di dati. DecimalSeparator specifica il separatore decimale utilizzato in Excel per il riconoscimento dei numeri ed infine ThousandsSeparator specifica il separatore delle migliaia utilizzato in Excel per il riconoscimento dei numeri.  
 Adesso che i parametri sono stati analizzati, vediamo come importare nel modo più semplice i dati del file di testo tags.txt in un nuovo foglio di calcolo.  
 Siccome i parametri sono tanti e la possibilità di confusione è elevata, vediamo prima come richiamare il metodo esplicitando i singoli parametri:

```
appExcel.Workbooks.OpenText FileName:="C:\Documenti\tags.txt",
Data Type:=xlDelimited, tab:=True
```

utilizzando invece la notazione tradizionale:

```
appExcel.Workbooks.OpenText "C:\Documenti\tags.txt", , , xlDelimited, , , True
```

In entrambi i casi il risultato sarà il seguente:



Lo stesso ragionamento fatto per il metodo OpenText può essere esteso ad un secondo metodo ossia **TextToColumns** che consente una semplice gestione delle celle che contengono un testo esteso su più colonne. L'esempio grafico riportato sotto mostra il caso in questione:

	A	B	C	D	E	F
1						
2	Il testo della cella è disposto su cinque colonne					
3	Il testo della cella è disposto su quattro colonne					
4	Il testo della cella è disposto su tre colonne					
5	Il testo della cella è disposto su due colonne					
6						
7						
8						

La sintassi prevista del metodo è del tutto simile a quella del metodo visto in precedenza:

`Range.TextToColumns(Destination, DataType, TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma, Space, Other, OtherChar, FieldInfo, DecimalSeparator, ThousandsSeparator)`

con due differenza fondamentali: la prima è che il metodo si riferisce ad un oggetto Range, ossia ad un gruppo di più celle.

La seconda è costituita dal parametro Destination che specifica la destinazione della separazione del testo che supera eventualmente il limite della colonna.

Tutti gli altri parametri hanno lo stesso significato sia nel caso di OpenText che nel caso di TextToColumns. Proviamo a fare un esempio veloce. Copiamo negli Appunti un testo qualsiasi formato da una o più frasi. Dopodichè diamo l'avvio al seguente codice (che naturalmente deve essere integrato con l'inizializzazione degli oggetti 'applicazione', 'foglio di lavoro', 'foglio di calcolo', come visto nei primi articoli)

```
foglioExcel.Activate
foglioExcel.Paste
```

Il risultato sarà che tutto il testo viene trasportato dagli Appunti al foglio di calcolo in tante righe quante sono le righe del testo, occupando quindi (se il testo è sufficientemente lungo) più dello spazio delimitato dalla colonna:

	A	B	C	D	E	F	G
1	Value indicating the aspect of the object whose limit is to be						
2	retrieved; the value is obtained from the enumerations DVASPECT						
3	and from DVASPECT2. Note that newer objects and containers						
4	that support optimized drawing interfaces support the DVASPECT2						
5	enumeration values. Older objects and containers that do not						
6	support optimized drawing interfaces may not support DVASPECT2.						
7	The most common value for this method is DVASPECT_CONTENT, which						
8	specifies a full rendering of the object within its container.						
9							

Adesso invece proviamo ad utilizzare il metodo *TextToColumns* indicando come separatore per ogni singola cella lo spazio (parametro *Space* impostato su *True*):

```
foglioExcel.Activate
foglioExcel.Paste
Selection.TextToColumns , xlDelimited, , True, , , , True
```

e come si può vedere dal risultato:

	A	B	C	D	E	F	G	H	I	J	K	L
1	Value	indicating	the	aspect	of	the	object	whose	limit	is	to	be
2	retrieved;	the	value	is	obtained	from	the	enumeratic	DVASPECT			
3	and	from	DVASPEC	Note	that	newer	objects	and	containers			
4	that	support	optimized	drawing	interfaces	support	the	DVASPECT2				
5	enumeratic	values.	Older	objects	and	containers	that	do	not			
6	support	optimized	drawing	interfaces	may	not	support	DVASPECT2.				
7	The	most	common	value	for	this	method	is	DVASPEC	which		
8	specifies	a	full	rendering	of	the	object	within	its	container.		
9												

tutte le righe del testo vengono suddivise in tante celle quante sono le parole con una parola per cella.

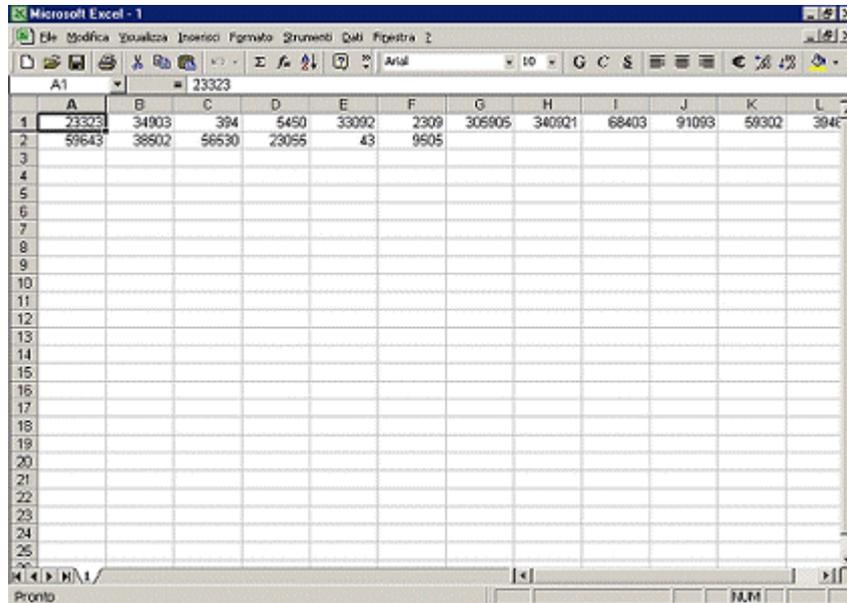
Naturalmente i metodi di suddivisione possono essere i più svariati, indicando ad esempio di suddividere il testo dopo ogni virgoletta, ogni punto e virgola e ogni qualsiasi altro carattere personalizzato.



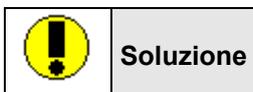
1) aprire ed attivare una nuova sessione, una cartella ed un foglio di calcolo Excel.

2) creare un file di testo denominato 1.txt ed aggiungervi il seguente testo:  
23323,34903,394,5450,33092,2309,305905,340921,68403,91093,59302,39466,  
59643,38502,56530,23055,43,9505

3) importare il testo di 1.txt in un foglio di calcolo e suddividere il testo usando come delimitatore la virgola. Ecco un esempio di come può essere impostata l'applicazione:



	A	B	C	D	E	F	G	H	I	J	K	L
1	23323	34903	394	5450	33092	2309	305905	340921	68403	91093	59302	39466
2	59643	38502	56530	23055	43	9505						
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												
23												
24												
25												



```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
appExcel.Workbooks.OpenText "C:\Documenti\1.txt", , , xlDelimited, , , , True
End Sub
```

## Le visualizzazioni

Un importante aspetto dell'utilizzo di applicazioni orientate a Microsoft Excel è la possibilità di un elevato livello di interazione con l'utente.

Per questo motivo può risultare utile visualizzare l'intera cartella di lavoro in modalità Web ossia proprio come se fosse una pagina creata in HTML.

Per far ciò è necessario utilizzare il metodo **WebPagePreview** la cui sintassi rispetta la seguente forma:

```
cartExcel.WebPagePreview
```

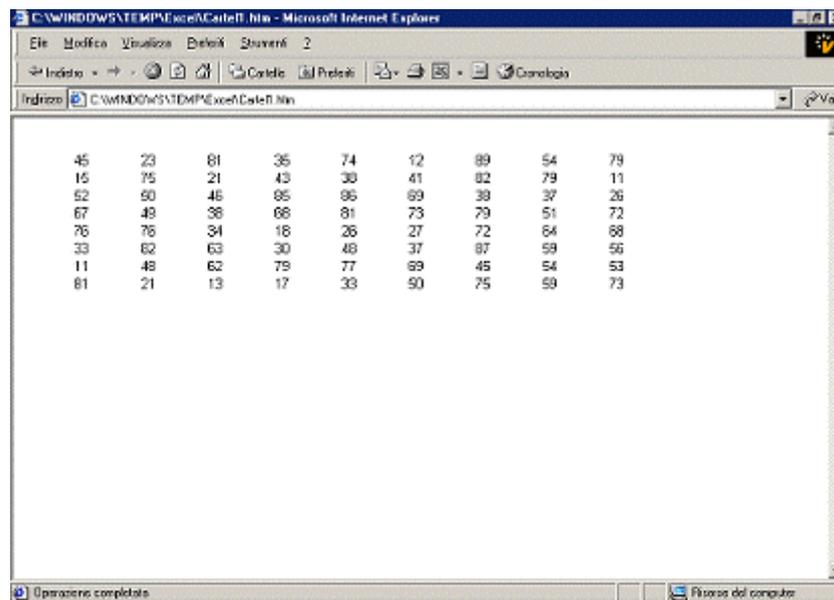
naturalmente il metodo è applicabile solamente ad una cartella di lavoro Excel e non ad esempio all'oggetto 'applicazione' nè ad un singolo foglio di calcolo. Se ad esempio popoliamo un Range con numeri casuali come già è stato visto e richiamiamo il metodo WebPagePreview, nel seguente modo:

```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
Dim Colonna1, Colonna2, Riga1, Riga2 As Integer
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
foglioExcel.Range("A2", "I9").Formula = "=INT(RAND()* (90-10)+10) "
cartExcel.WebPagePreview
End Sub

```

il foglio di lavoro presenterà un *Range* A2,I9 popolato di numeri casuali compresi tra 10 e 90. Una volta richiamato il metodo *WebPagePreview* (in questo caso quindi all'apertura del foglio di lavoro) la pagina Web visualizzata sarà la seguente:



Come si può notare la pagina viene creata come file temporaneo nella cartella "TEMP" all'interno della cartella di sistema del proprio computer, ed ha lo stesso nome della cartella di lavoro con la quale si sta operando.

## Operazioni tra celle

Un altro metodo interessante è **ColumnDifferences** che calcola le differenze tra le celle di una colonna e una cella di riferimento.

Quindi questo metodo rappresenta un confronto solo tra le celle di una singola colonna. Ad esempio se assegnamo alle celle della colonna A dei valori che vanno da 1 a 5 per le prime cinque celle, da 7 a 13 per le seconde 7 ed assegnamo il valore 1 alla sesta cella, possiamo determinare ad esempio quali celle sono differenti dalla cella A1 di valore 1. Il codice seguente ha proprio tale finalità:

```

foglioExcel.Activate
For i = 1 To 5
foglioExcel.Cells(i, 1).Value = i
Next i
foglioExcel.Cells(6, 1).Value = 1
For i = 7 To 13
foglioExcel.Cells(i, 1).Value = i
Next i
Set Differenza = foglioExcel.Columns("A").ColumnDifferences( _

```

```
Comparison:=ActiveSheet.Range("A1")
Differenza.Select
```

Com'è facile immaginare le celle selezionate saranno quelle di valore diverso da 1 ossia:

	A	B	C	D	E	F
1	1					
2	2					
3	3					
4	4					
5	5					
6	1					
7	7					
8	8					
9	9					
10	10					
11	11					
12	12					
13	13					

## La curva di Bézier

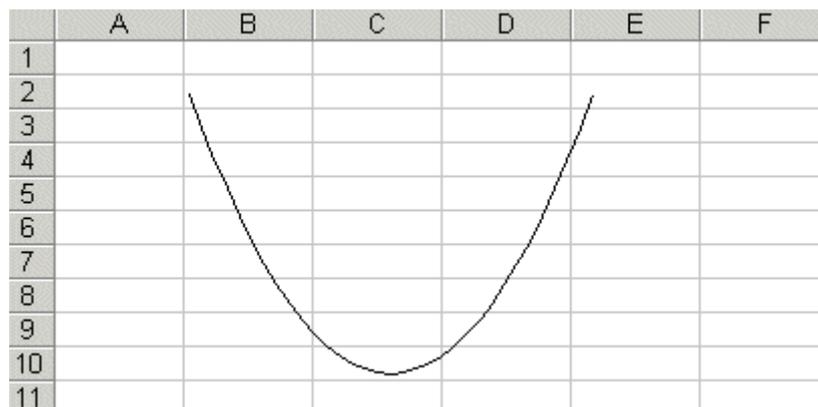
Esiste poi un metodo particolare che permette di creare una **curva di Bézier** che consente di disegnare una curva a mano libera ma con maggior precisione in quanto una volta definiti dall'utente i punti intermedi, l'applicazione ne calcola i punti intermedi. Una curva di Bézier rappresenta un oggetto **Shape** di cui si parlerà in maniera più approfondita più in là con gli articoli.

La sintassi del metodo **AddCurve** di cui si parla è la seguente:

```
foglioExcel.AddCurve(SafeArrayOfPoints)
```

dove l'unico parametro da indicare ossia *SafeArrayOfPoints* è la matrice di coppie di coordinate (x,y) che rappresentano i punti intermedi della curva.

Il primo punto da specificare è il vertice di partenza mentre l'ultimo punto da specificare è il vertice di arrivo della curva. Ad esempio consideriamo una curva come quella mostrata in figura:



il codice da utilizzare sarà il seguente:

La sintassi del metodo **AddCurve** di cui si parla è la seguente:

```
foglioExcel.Activate
Dim pts(1 To 4, 1 To 2) As Single
pts(1, 1) = 50 'punto 1 coordinata X
pts(1, 2) = 20 'punto 1 coordinata Y
pts(2, 1) = 100 'punto 2 coordinata X
pts(2, 2) = 160 'punto 2 coordinata Y
```

```

pts(3, 1) = 150 'punto 3 coordinata X
pts(3, 2) = 160 'punto 3 coordinata Y
pts(4, 1) = 200 'punto 4 coordinata X
pts(4, 2) = 20 'punto 4 coordinata Y
foglioExcel.Shapes.AddCurve pts

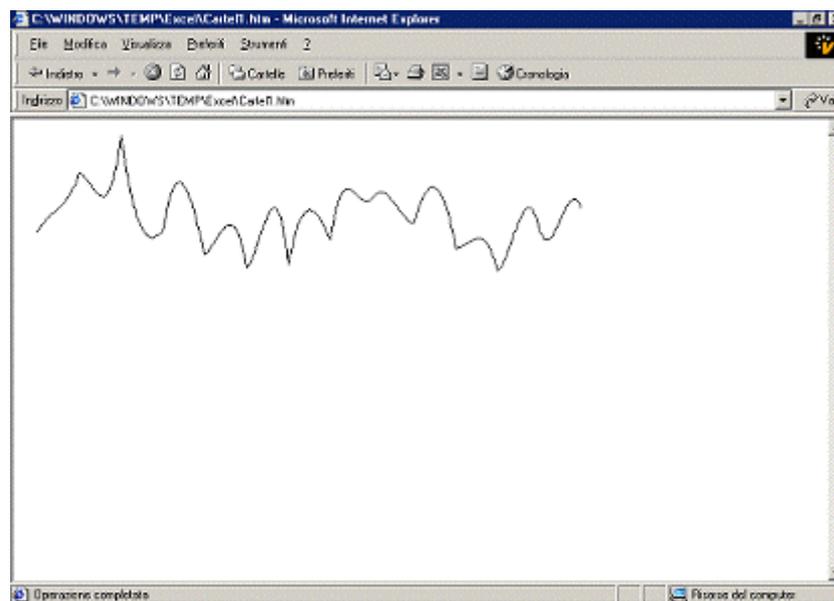
```

In pratica abbiamo dichiarato la matrice pts. La notazione 1 To 4 indica che si vogliono memorizzare in tale matrice quattro punti della curva mentre la notazione 1 To 2 indica invece che di questi quattro punti si vuole specificare le due coordinate X ed Y. Naturalmente sviluppare un grafico utilizzando una *curva di Bézier* è molto macchinoso ma può risultare divertente.

Nell'esercizio proposto al termine del capitolo si proverà a tradurre in grafico (o meglio in curva) dei numeri casuali.



- 1) aprire ed attivare una nuova sessione, una cartella ed un foglio di calcolo Excel.
  - 2) creare una curva di Bèzier con 40 punti le cui coordinate X vengono incrementate di 10 da punto a punto mentre le coordinate Y sono rappresentate per ogni punto da un numero casuale compreso nell'intervallo (0,100);
  - 3) esportare la curva in una pagina Web e visualizzarla;
- Ecco un esempio di come può essere impostata l'applicazione:



```

Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = True
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate

```

```

Dim pts(1 To 40, 1 To 2) As Single
For i = 1 To 40
pts(i, 1) = i * 10
pts(i, 2) = Int(100 * Rnd)
Next i
foglioExcel.Shapes.AddCurve pts
End Sub

```

## Importazione di un foglio di calcolo Excel tramite un contenitore OLE

Come abbiamo potuto notare interagire con una sessione di Excel non va oltre le normali difficoltà del linguaggio, ma come ci si deve comportare se si desidera inserire un foglio di calcolo Excel nel proprio progetto?

Può infatti risultare scomodo dover aprire un'applicazione esterna alla propria (tempi di caricamento, focus che passa da una finestra all'altra e così via).

Vediamo quindi in questo capitolo (che rappresenta una sorta di parentesi nel lungo discorso che stiamo affrontando) come utilizzare un **contenitore OLE**.

OLE infatti non è altro che un recipiente dove inserire un oggetto di una classe qualsiasi. La classe dell'oggetto inseribile è definito dalla proprietà **Class**:

```
OLE.Class = nome_classe
```

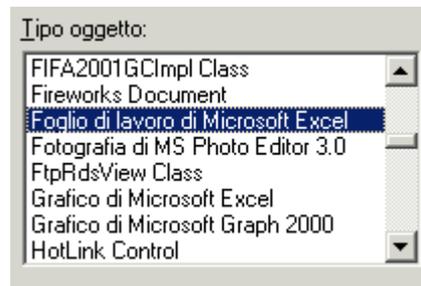
ed è rappresentata dal tipo di oggetto che si può importare nel contenitore. Alcune di queste classi sono: documento di testo, cartella Excel, foglio di calcolo Excel, documento Word, immagine bitmap, immagine jpeg e così via.

Non è necessario tuttavia indicare il nome della classe perché quando si indica il documento da importare, Visual Basic riconosce automaticamente a quale classe appartiene.

Prima di continuare vediamo quindi i passi preliminari. Prima di tutto si dovrà importare sul piano il controllo OLE.

Una volta trascinato sul piano si aprirà una finestra nella quale scegliere il tipo di oggetto (o meglio la classe dell'oggetto) importabile nel contenitore OLE.

Lavorando con i fogli di calcolo Excel, dovremo quindi scegliere la voce 'Foglio di lavoro di Microsoft Excel' come mostrato nella figura sotto:



Adesso tutto è pronto per poter lavorare. La primissima cosa da fare è quella di importare un foglio di lavoro Excel esistente attraverso il metodo **CreateLink**.

Tale metodo infatti permette di memorizzare e gestire effettivamente i dati dell'oggetto incorporato (il foglio di lavoro Excel) dall'applicazione in cui sono stati creati.

```

Private Sub Form_Load()
OLE1.CreateLink "C:\Documenti\VBALIST.xls"
End Sub

```

Dando già un primo avvio all'applicazione il foglio di calcolo dovrebbe essere visualizzato in questo modo:

Parole chiave di VBA	
Inglese	Italiano
Abs	Abs
Access	Accesso
Alias	Alias
And	And
Any	Qualsiasi
AppActivate	AttivaApp
Append	Accodamento
ArgList	ElencoArgomenti
ArgName	Espressione
Array	Matrice
.	.

Naturalmente le dimensioni della sezione del foglio visualizzata dipendono da quelle del contenitore OLE. In ogni caso è possibile effettuare un ingrandimento attraverso la proprietà **SizeMode** alla quale assegnare una delle seguenti costanti:

Costante	Risultato ottenuto
VbOLESizeClip	il foglio di calcolo viene visualizzato su scala 1:1
VbOLESizeStretch	il foglio di calcolo viene visualizzato in modo da occupare tutta la superficie del contenitore OLE;
VbOLESizeAutoSize	il contenitore OLE prende le dimensioni del foglio di calcolo;
VbOLESizeZoom	il foglio di calcolo viene ingrandito in modo da mantenere però le dimensioni originali;

Purtroppo però le operazioni possibili con il controllo OLE sono molto più limitate rispetto a quelle possibili utilizzando gli oggetti Excel.Application, Excel.Workbook ed Excel.Worksheet però nulla impedisce di rendere invisibile l'applicazione Excel e di visualizzare le modifiche compiute via via all'interno del contenitore OLE.

Ad esempio, se prendiamo come riferimento il codice visto nello scorso capitolo, quello cioè che permetteva la creazione di una curva di Bèzier, possiamo adattare il progetto in modo da rendere invisibile la sessione di lavoro Excel.Application (indicata come appExcel):

```
Dim appExcel As New Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet
Private Sub Form_Load()
appExcel.Visible = False
Set cartExcel = Excel.Workbooks.Add
Set foglioExcel = Excel.Worksheets.Item(1)
foglioExcel.Activate
```

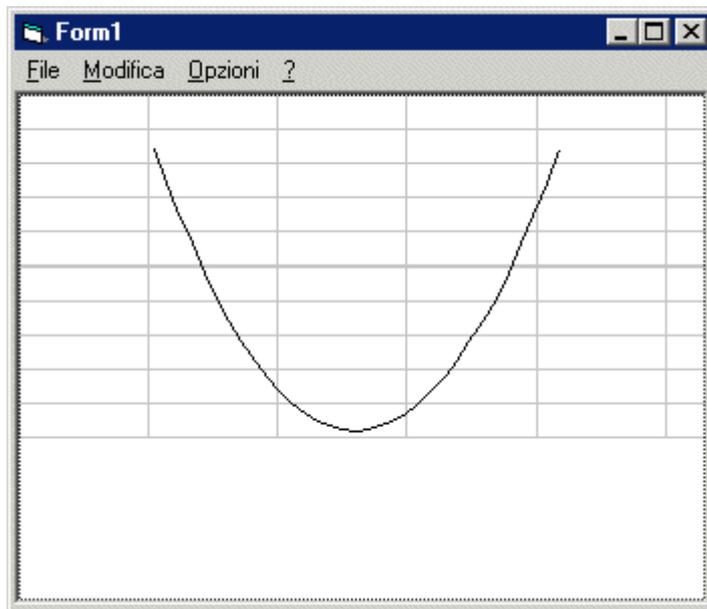
quindi creare la curva di Bèzier:

```
Dim pts(1 To 4, 1 To 2) As Single
pts(1, 1) = 50
pts(1, 2) = 20
pts(2, 1) = 100
pts(2, 2) = 160
pts(3, 1) = 150
pts(3, 2) = 160
pts(4, 1) = 200
pts(4, 2) = 20
foglioExcel.Shapes.AddCurve pts
```

e quindi collegare il foglio di lavoro (che dev'essere esistente, quindi precedentemente salvato sul disco fisso) al contenitore OLE:

```
OLE1.CreateLink "C:\Documenti\bezier.xls"
```

questo punto il contenitore OLE visualizzerà la curva in questo modo:



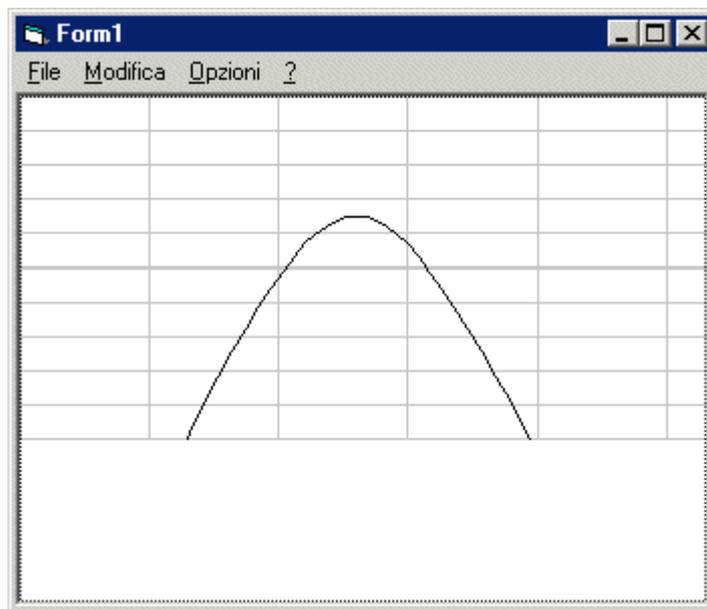
Che cosa accade però nel caso in cui s'impostino i punti in maniera differente? Si consideri il seguente caso:

```
Dim pts(1 To 4, 1 To 2) As Single  
pts(1, 1) = 50  
pts(1, 2) = 150  
pts(2, 1) = 120  
pts(2, 2) = 10  
pts(3, 1) = 130  
pts(3, 2) = 10  
pts(4, 1) = 200  
pts(4, 2) = 150  
foglioExcel.Shapes.AddCurve pts
```

dando l'avvio all'applicazione si visualizzerà sempre la curva mostrata sopra mentre i punti indicati in quest'ultimo codice ruotano la curva sull'asse orizzontale. Sarà quindi necessario creare un collegamento ad un file .xls aggiornato. E l'aggiornamento del file "bezier.xls" è possibile semplicemente con un salvataggio. Se infatti si aggiunge prima di richiamare il metodo CreateLink l'istruzione di salvataggio ed eventualmente si compie un aggiornamento del contenuto del controllo OLE attraverso il metodo **Refresh**:

```
foglioExcel.SaveAs "bezier.xls"  
OLE1.Refresh
```

il contenitore potrà visualizzare la nuova curva, in questo modo:



## Riempire le celle con un solo metodo

E' importante poter avere il pieno controllo di tutti i dati presenti sul foglio di calcolo. Ad esempio si può avere la necessità di riempire oppure copiare il *Range* specificato a partire dalla cella indicata o dalle celle superiori, fino a quelle inferiori.

Il metodo da utilizzare in questo caso è **FillDown**. In questo modo tra le altre cose, viene evitato di ripetere lunghi cicli per riempire le celle. Facciamo un esempio. Se da codice si volesse inserire nelle celle A1, A2, A3, A4, A5, A6, A7, A8, A9 e A10 lo stesso valore (ad esempio il testo "cella") si dovrà agire in questo modo: inserire il testo nella prima cella, A1 e richiamare il metodo FillDown per le celle A1-A10.

Perchè partire proprio dalla cella A1 visto che le è già stato assegnato il valore in modo 'manuale'? La ragione è che, come già accennato prima, il metodo FillDown individua un Range (nel caso in esame di celle A1,A10) la cui prima cella è quella da copiare e riprodurre nelle seguenti.

Esempio:

```
foglioExcel.Cells(1, 1).Value = "cella"
foglioExcel.Range("A1:A10").FillDown
```

Mentre con la prima linea si imposta il valore della cella A1 su "cella", con la seconda si copia tale valore nelle celle ad essa successive, ossia A2, A3, ... , A10.

Il risultato da un punto di vista grafico è il seguente:

	A	B	C	D	E	F
1	cella					
2	cella					
3	cella					
4	cella					
5	cella					
6	cella					
7	cella					
8	cella					
9	cella					
10	cella					

Che cosa accade se si richiama il metodo *FillDown* per un *Range* che va al di là di una singola colonna, indicando ad esempio il seguente codice:

```
foglioExcel.Range("A1:B10").FillDown
```

Specificando la linea di codice sopra si indica che le celle A2, A10 devono essere riempite col valore della cella A1 e che le celle B2, B10 con quello della cella B1. Perciò avviando l'applicazione in questo modo si otterrà lo stesso risultato di prima perchè alla cella B1 non è stato assegnato alcun valore. In realtà le celle B2, B10 verranno riempite ugualmente, ma col valore della cella B1 ossia "".

Per ottenere un risultato si dovrà indicare espressamente il valore della cella B2, ad esempio:

```
foglioExcel.Cells(1, 2).Value = "colonna 2"
```

il risultato a questo punto sarà il seguente:

	A	B	C	D	E	F
1	colonna 1	colonna 2				
2	colonna 1	colonna 2				
3	colonna 1	colonna 2				
4	colonna 1	colonna 2				
5	colonna 1	colonna 2				
6	colonna 1	colonna 2				
7	colonna 1	colonna 2				
8	colonna 1	colonna 2				
9	colonna 1	colonna 2				
10	colonna 1	colonna 2				

naturalmente indicando "colonna 1" e non più "cella" come valore per il Range A1,A10. Per riempire le celle in ogni direzione e non solo in verticale, esiste una serie di metodi alternativi. Uno di essi è **FillLeft**, che come s'intuisce, assegna il valore della cella indicata a quelle alla sua sinistra, fino alla fine del Range. Ad esempio, partendo dalla cella F1 ossia dalla cella di coordinate 1 (riga), 6 (colonna), si può provare a riempire tutte le celle alla sua sinistra, fino alla A1:

```
foglioExcel.Cells(1, 6).Value = "sinistra"
foglioExcel.Range("F1:A1").FillLeft
```

ottenendo così il seguente effetto:

	A	B	C	D	E	F
1	sinistra	sinistra	sinistra	sinistra	sinistra	sinistra
2						
3						
4						
5						
6						
7						
8						
9						
10						

E' importante notare l'assegnazione del Range in questo secondo caso. Non è possibile indicare una selezione di celle A1,F1 anche se in altre occasioni questo potrebbe essere indifferente. La spiegazione è che, come per FillDown, la prima cella del Range indica il valore da copiare.

Indicando così F1,A1 si copia la cella F1 in E1, D1, C1, B1 e A1, mentre indicando il Range A1,F1 si sarebbe copiata la cella A1.

Anche in questo caso valgono le considerazioni fatte per il caso precedente: se si indica un Range di più colonne si dovrà indicare la prima cella di ogni colonna.

Molto simile ai metodi appena analizzati è **FillRight** che riempie le celle alla destra della cella di partenza, entro il Range specificato.

Ad esempio se si vuole ottenere un risultato simile al precedente, partendo però dalla cella A1 e non più dalla F1, bisognerà indicare il valore della cella A1 e richiamare il metodo FillRight specificando il Range A1,F1 (e quindi non più F1,A1):

```
foglioExcel.Cells(1, 1).Value = "destra"  
foglioExcel.Range("A1:F1").FillRight
```

Il logico risultato in questo caso è il seguente:

	A	B	C	D	E	F
1	destra	destra	destra	destra	destra	destra
2						
3						
4						
5						
6						
7						
8						
9						
10						

Per terminare le considerazioni su questa serie di metodi è necessario analizzare anche **FillUp** che assegna il valore della cella indicata a tutte le celle al di sopra di essa.

Se infatti si prende in considerazione la cella A10 e si vuole riempire il Range A10,A1 (partendo sempre dalla cella a cui è stato assegnato un valore ossia in questo caso A10):

```
foglioExcel.Cells(10, 1).Value = "sopra"  
foglioExcel.Range("A10:A1").FillUp
```

Si otterrà quindi un risultato molto simile al primo analizzato. In questo caso però il punto di partenza non è la cella A1 ma la cella A10 ed il verso dell'operazione non è dall'alto verso il basso ma dal basso verso l'alto:

	A	B	C	D	E	F
1	sopra					
2	sopra					
3	sopra					
4	sopra					
5	sopra					
6	sopra					
7	sopra					
8	sopra					
9	sopra					
10	sopra					



1) aprire ed attivare una nuova sessione, una cartella ed un foglio di calcolo Excel.

2) all'avvio dell'applicazione selezionare il primo foglio di lavoro, renderlo attivo e popolare il Range A1,F10 col testo "cella" utilizzando il minor numero di volte (ne sono sufficienti due) i metodi FillDown, FillUp, FillRight e FillLeft.

Ecco un esempio di come può essere impostata l'applicazione:

	A	B	C	D	E	F
1	cella	cella	cella	cella	cella	cella
2	cella	cella	cella	cella	cella	cella
3	cella	cella	cella	cella	cella	cella
4	cella	cella	cella	cella	cella	cella
5	cella	cella	cella	cella	cella	cella
6	cella	cella	cella	cella	cella	cella
7	cella	cella	cella	cella	cella	cella
8	cella	cella	cella	cella	cella	cella
9	cella	cella	cella	cella	cella	cella
10	cella	cella	cella	cella	cella	cella

## L'oggetto Scenario e l'insieme Scenarios

L'oggetto Scenario, facente parte dell'insieme **Scenarios** e legato all'oggetto WorkSheet rappresenta uno scenario in un foglio di lavoro, ovvero un gruppo di valori detti celle variabili.

Uno Scenario è un insieme di valori che possono essere sostituiti automaticamente all'interno del foglio di lavoro e rappresenta un utile strumento (denominato "metodo con analisi what-if") per prevedere i possibili risultati di un foglio di lavoro: si può ad esempio considerare diversi gruppi di celle (più Range) di un foglio di lavoro, e quindi considerare ipotesi differenti passando a tali scenari per visualizzare i diversi risultati.

L'esempio riportato dalla guida in linea di MS Excel è esplicativo:

"[...] Se ad esempio si desidera creare un bilancio, ma le entrate non sono certe, è possibile definire diversi valori delle entrate e passare da uno scenario all'altro per effettuare un'analisi per ipotesi.

	A	B
1	Entrate lordi	L. 5.000.000
2	Costo dei beni venduti	L. 1.320.000
3	Utile lordo	L. 3.620.000

	A	B
1	Entrate lordi	L. 15.000.000
2	Costo dei beni venduti	L. 2.600.000
3	Utile lordo	L. 12.400.000

È possibile denominare lo scenario dell'esempio precedente "Caso peggiore", impostare il valore nella cella B1 a L. 5.000.000 e il valore nella cella B2 a L. 1.320.000. Il nome del secondo scenario potrebbe essere "Caso migliore", il valore in B1 L. 15.000.000 e il valore in B2 L. 2.600.000. [...]"

All'interno dell'insieme Scenarios (definito oggetto-insieme in quanto anch'esso rappresenta un'oggetto a sé stante), ogni Scenario è dotato di indice.

Per indicare così un particolare Scenario si indicherà:

```
Scenarios(indice_scenario)
```

dove `indice_scenario` rappresenta proprio l'indice dello Scenario.

Per poter creare un nuovo Scenario per il foglio di lavoro (foglioExcel) correntemente utilizzato ed aggiungerlo col relativo indice all'insieme Scenarios, si utilizza il metodo `Add`, la cui sintassi è la seguente:

```
FoglioExcel.Scenarios.Add(nome, celle_variabili, valori, commento, bloccato, nascosto)
```

dove `nome` indica naturalmente il nome in formato String che si desidera assegnare al nuovo scenario (il nome dello Scenario deve essere unico, quindi se si tenta di creare uno Scenario con un nome già in uso si avrà un errore) `celle_variabili` è un valore di tipo Variant che indica il Range ossia il gruppo di celle da associare all'oggetto Scenario, `valori` indica invece una matrice che contenga i valori da assegnare alle celle del Range indicato.

Se valori viene omissi allora si utilizzeranno i valori correnti contenuti correntemente all'interno delle celle indicate.

Commento è invece un dato di tipo String che indica un commento personalizzato che è possibile assegnare al particolare Scenario di nuova creazione.

Se viene indicato un commento valido, quando viene aggiunto un nuovo scenario il nome dell'autore e la data vengono automaticamente aggiunti all'inizio del relativo commento.

Bloccato indica se lo Scenario verrà bloccato (True) o meno (False) per permettere o impedire a qualsiasi utente del foglio di lavoro di modificare i valori dello Scenario stesso.

Infine nascosto indica se lo Scenario verrà mantenuto invisibile (True) o meno (False).

Per comprendere l'utilità del metodo si consideri il seguente esempio:

in un foglio di calcolo sono stati immessi i valori relativi a diversi accessori di computer:

	A	B	C
1	Componente	Costo in €	
2	RAM 128MB	78	
3	RAM 512MB	258	
4	RAM 256MB	141	
5	HD 20GB	165	
6	CD-ROM 48X	75	
7	Monitor 17"	240	
8			

[www.vbitalia.it](http://www.vbitalia.it)

La tabella sopra può essere generata nel seguente modo:

```
With foglioExcel
```

```
.Cells(1, 1).Value = "Componente": .Cells(1, 2).Value = "Prezzo in €"
```

```
.Cells(2, 1).Value = "RAM 128MB": .Cells(2, 2).Value = "78"
```

```
.Cells(3, 1).Value = "RAM 512MB": .Cells(3, 2).Value = "258"
```

```
.Cells(4, 1).Value = "RAM 256MB": .Cells(4, 2).Value = "141"
```

```
.Cells(5, 1).Value = "HD 20GB": .Cells(5, 2).Value = "165"
```

```
.Cells(6, 1).Value = "CD-ROM 48X": .Cells(6, 2).Value = "75"
```

```
.Cells(7, 1).Value = "Monitor 17'1": .Cells(7, 2).Value = "240"
```

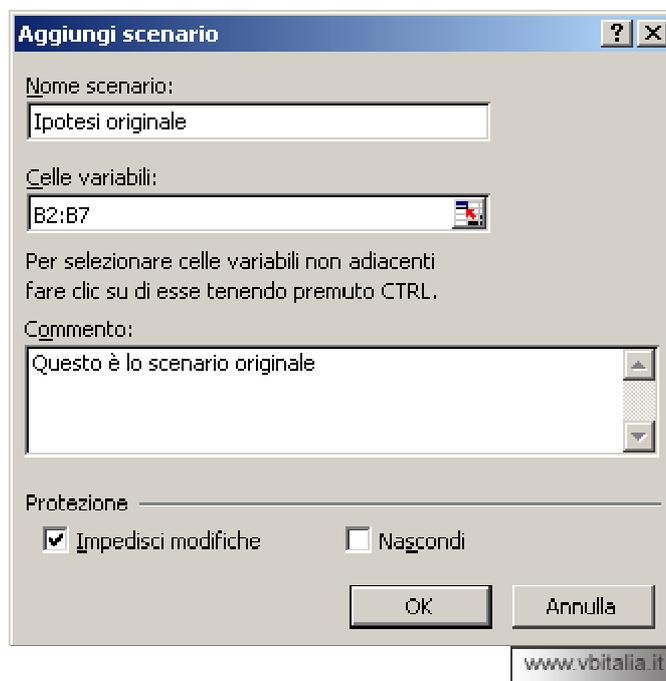
```
End With
```

Siccome questi sono i dati originali, salviamoli come primo scenario in quanto successivamente le celle verranno modificate automaticamente. Si applicherà dunque il metodo **Add** alle celle dei valori numerici ossia B2-B7 assegnando allo Scenario il nome "Ipotesi originale" ed il commento "Questo è lo scenario originale":

```
foglioExcel.Scenarios.Add "Ipotesi originale", foglioExcel.Range("B2:B7"), _  
"Questo è lo scenario originale.", True
```

Con il valore True si indica il blocco dello scenario da eventuali modifiche.

Il codice appena visto corrisponde all'esecuzione della finestra Excel "Aggiungi scenario", raggiungibile dal menu Strumenti >> Scenari:



Adesso è possibile fare una prima prova contando gli Scenari presenti nell'insieme Scenarios utilizzando così il metodo Count dell'insieme-oggetto Scenarios:

```
MsgBox foglioExcel.Scenarios.Count
```

Il risultato sarà naturalmente 1.

Ora utilizzando le proprietà Name, Comment, Index, Hidden e Locked si può determinare le proprietà di tale Scenario, proprietà che erano state impostate al momento della creazione:

```
Totale_Scenari = foglioExcel.Scenarios.Count
For ii = 1 To Totale_Scenari
MsgBox "Nome: " & foglioExcel.Scenarios(ii).Name & vbCrLf & _
"Commento: " & foglioExcel.Scenarios(ii).Comment & vbCrLf & _
"Nascosto? " & foglioExcel.Scenarios(ii).Hidden & vbCrLf & _
"Indice: " & foglioExcel.Scenarios(ii).Index & vbCrLf & _
"Bloccato? " & foglioExcel.Scenarios(ii).Locked & vbCrLf
Next ii
```



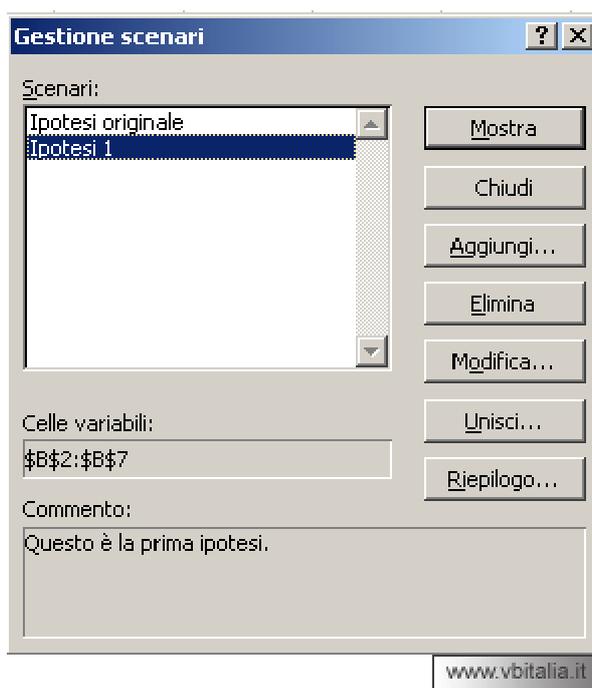
Queste sono solo alcune delle proprietà di cui l'oggetto Scenario dispone, le uniche che possano essere visualizzate in una finestra di messaggio. In aggiunta esistono ulteriori proprietà tra cui le più significative sono: **ChangingCells** che come visto poco fa restituisce un oggetto Range con le celle variabili in esso contenute, la proprietà **Values** che restituisce una matrice contenente i valori correnti delle celle variabili dello scenario, la proprietà **Creator** (che comunque sarebbe stato possibile visualizzare nella finestra di messaggio) che restituisce l'applicazione che ha creato l'oggetto, sottoforma di intero a 32 bit. Ad esempio se l'oggetto è stato creato da MS Excel come nel caso preso in esame, si otterrà stringa XCEL, corrispondente al valore esadecimale 5843454C.

Si provi adesso a creare un secondo Scenario utilizzando valori differenti da quelli indicati poco fa come originali ad esempio valori 75, 250, 153, 82, 260 come componenti di una matrice di valori, ed associare al nuovo Scenario il nome "Ipotesi 1" ed il commento "Questa è la prima ipotesi".  
Il valore di blocco dello Scenario sarà anche questa volta impostato su True:

```
foglioExcel.Scenarios.Add "Ipotesi 1", foglioExcel.Range("B2:B7"), _  
Array(75, 250, 153, 82, 260), "Questo è la prima ipotesi.", True
```

Si noterà che è stato utilizzato lo stesso Range dello Scenario originale (B2:B7) per permettere ai valori di occupare le caselle dove nell'ipotesi originale erano contenuti i prezzi in euro. Una seconda cosa da notare è che gli elementi della matrice sono 5 e non 6 come i componenti nella lista perché si suppone che ad esempio il costo dell'ultimo elemento, il monitor, sia certo.

Ora si riapra nel foglio di lavoro la finestra "Gestione scenari": cliccando su "Riepilogo..." si otterrà la lista degli scenari creati:



Si crei un terzo scenario inserendo questa volta i valori 60, 278, 139, 64, 250, inclusi sempre in una matrice di valori ed assegnandogli il nome "Ipotesi 2", il commento "Questa è la seconda ipotesi" e impostando la proprietà **Locked** su True:

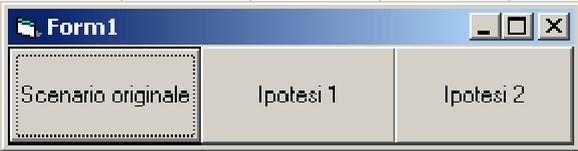
```
foglioExcel.Scenarios.Add "Ipotesi 2", foglioExcel.Range("B2:B7"), _  
Array(60, 278, 139, 64, 250), "Questo è la seconda ipotesi.", True
```

Ora si può passare agilmente da uno Scenario all'altro semplicemente utilizzando il metodo Show dell'oggetto Scenarios ed indicando l'indice dello Scenario da visualizzare (1 per "Ipotesi originale", 2 per "Ipotesi 1" e 3 per "Ipotesi 2").

Se si inserisce nel progetto un oggetto CommandButton per ogni Scenario si può passare da uno Scenario all'altro con la pressione del tasto corrispondente:

```
Private Sub Command1_Click()  
foglioExcel.Scenarios(1).Show  
End Sub  
Private Sub Command2_Click()  
foglioExcel.Scenarios(2).Show  
End Sub  
Private Sub Command3_Click()  
foglioExcel.Scenarios(3).Show  
End Sub
```

Questo è lo Scenario visualizzato alla pressione di Command1:

	A	B	C	D	E
1	Componenti	Prezzo in €			
2	RAM 128M	78			
3	RAM 512M	258			
4	RAM 256M	141			
5	HD 20GB	165			
6	CD-ROM 4	75			
7	Monitor 17"	240			
8					
9					
10					
11					
12					

www.vbitalia.it

questo è lo Scenario visualizzato premendo Command2:

	A	B	C	D	E
1	Componenti	Prezzo in €			
2	RAM 128M	75			
3	RAM 512M	250			
4	RAM 256M	153			
5	HD 20GB	82			
6	CD-ROM 4	260			
7	Monitor 17"	240			
8					
9					
10					
11					
12					

www.vbitalia.it

e questo è lo Scenario visualizzato premendo Command3:

	A	B	C	D	E
1	Componenti	Prezzo in €			
2	RAM 128M	60			
3	RAM 512M	278			
4	RAM 256M	139			
5	HD 20GB	64			
6	CD-ROM 4	250			
7	Monitor 17"	240			
8					
9					
10					
11					
12					

www.vbitalia.it

Per finire si può creare un nuovo foglio di lavoro contenente un rapporto di riepilogo degli scenari presenti nel foglio di lavoro corrente attraverso il metodo CreateSummary. Si inserisca sul piano un quarto controllo CommandButton:

	A	B	C	D	E	F	G
1							
2		<b>Riepilogo scenari</b>					
3		Valori correnti: Ipotesi originale			Ipotesi 1		Ipotesi 2
5		<b>Celle variabili:</b>					
6		<b>\$B\$2</b>	78	78	75	60	
7		<b>\$B\$3</b>	258	258	250	278	
8		<b>\$B\$4</b>	141	141	153	139	
9		<b>\$B\$5</b>	165	165	82	64	
10		<b>\$B\$6</b>	75	75	260	250	
11		<b>\$B\$7</b>	240	240	240	240	
12							
13		Note: la colonna Valori correnti riporta i valori delle celle variabili nel momento in cui il Riepilogo scenari è stato creato. Le celle variabili sono evidenziate in grigio.					
14							
15							
16							
17							
18							
19							
20							
21							
22							

**Form1**

Scenario originale	Ipotesi 1	Ipotesi 2
--------------------	-----------	-----------

Riepilogo scenari
-------------------

[www.vbitalia.it](http://www.vbitalia.it)

La prima colonna indica le celle variabili, la seconda i valori correntemente visualizzati sul foglio di lavoro mentre tutte le successive rappresentano i valori contenuti nei rispettivi Scenari.

## L'oggetto Area e l'insieme Areas

Fa parte dell'oggetto Range, ossia di un gruppo di celle prese secondo una logica qualsiasi, anche l'oggetto Area. Per questo motivo quando ci si riferisce a tale oggetto e lo si richiama, verrà restituito sempre un Range.

La definizione di Area non è particolarmente intuitiva se non se ne illustra la funzione mediante un esempio: s'immagini quindi di avere un foglio di calcolo, anche senza alcun valore inserito nelle celle.

Selezionando un gruppo di celle si otterrà un Range, come già visto in precedenza. Nulla vieta però di considerare non soltanto una serie di celle contigue, ma più gruppi separati selezionabili contemporaneamente mantenendo premuto il tasto Ctrl nel momento della selezione.

L'unione di questi due gruppi forma un solo oggetto Range, ma ognuno di essi si può individuare come singolo oggetto Area.

La figura sottostante mostra l'esempio di due Areas che costituiscono un oggetto Range:

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

www.vbitalia.it

A questo punto, per verificare l'effettiva presenza di due aree, si può prevedere nel progetto una finestra di messaggio che alla pressione di un pulsante mostri il numero di oggetti Area presenti sul foglio di calcolo:

```
Private Sub Command1_Click()
MsgBox appExcel.Selection.Areas.Count
End Sub
```

Notare che Areas è stato legato all'oggetto Selection e non esplicitamente a Range. Questo per il semplice motivo che anche Selection costituisce un Range essendo il gruppo di celle (contigue o separate) selezionato dall'utente.

Inoltre si sarà notato che anche in questo caso il conto delle Aree presenti in un Range di selezione viene effettuata attraverso la proprietà Count.

Essendo ogni Area indipendente, se la si considera al di fuori dell'oggetto Range a cui appartiene, è possibile individuarla in modo univoco attraverso un indice, assegnatole in modo automatico al momento della creazione.

Riprendendo quindi la figura vista sopra, il gruppo di celle A1-B4 sarà individuato dall'indice 1 mentre il gruppo E1-E6 dall'indice 2, secondo una numerazione crescente che parte dal valore 1.

La possibilità di individuare ciascuna Area come oggetto a sé stante rappresenta un enorme vantaggio in quanto consente di considerare ogni Area come un singolo oggetto Range.

Se ne possono applicare pertanto i metodi e le proprietà legate a tale oggetto.

Ad esempio, si consideri una selezione formata da tre aree non contigue tra loro come in figura:

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							

www.vbitalia.it

Di ognuna di queste aree si vuole determinare il numero di celle di cui si compone, alla pressione di un pulsante CommandButton.

Per sviluppare questo progetto sarà necessario innanzitutto eseguire l'operazione per tutti gli oggetti Area presenti sul foglio di calcolo. A tale scopo si utilizzerà allora la proprietà Count citata poc'anzi:

```
Private Sub Command1_Click()
For i = 1 To appExcel.Selection.Areas.Count
' ...
' ...
' ...
Next i
```

End Sub

Il principio col quale scrivere la linea di codice che manca al blocco qui sopra è molto semplice: si sfrutterà la proprietà Count di Cells che fa riferimento al numero di celle presenti in un Range. E siccome un'Area costituisce proprio un Range, non si verificheranno problemi di proprietà non supportate dall'oggetto. La linea da inserire al posto dei commenti è la seguente:

```
MsgBox appExcel.Selection.Areas.Item(i).Cells.Count
```

In modo analogo, aumentando il numero delle operazioni da eseguire sulla selezione, è possibile determinare per ciascuna area il valore di ciascuna cella.

In questo caso però, oltre al ciclo che va a coprire tutte le aree esistenti, deve esserne previsto un secondo che vada a coprire tutte le celle all'interno di ogni Area.

Il codice mostrato di seguito esegue tale operazione:

```
For i = 1 To appExcel.Selection.Areas.Count
For ii = 1 To appExcel.Selection.Areas.Item(i).Cells.Count
MsgBox appExcel.Selection.Areas.Item(i).Cells(ii).Value
Next ii
Next i
```

---

## Riferimenti

Questo e-book è stato creato raggruppando le seguenti risorse:

<http://digilander.libero.it/VBItalia/Lezioni/excel1.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel2.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel3.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel4.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel5.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel6.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel7.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel8.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel9.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel10.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel11.htm>  
<http://digilander.libero.it/VBItalia/Lezioni/excel12.htm>  
<http://www.vbitalia.it/articoli/excel13.asp?lez=excel13>  
<http://www.vbitalia.it/articoli/excel14.asp?lez=excel14>