

# ***CREARE UN'AGGIUNTA A MICROSOFT EXCEL***

***Come collegare a Microsoft Excel add-  
in creati in Visual Basic***

***Visual Basic Italia***

## Indice

Premessa.....	3
Preparazione per la creazione di un'aggiunta.....	3
La creazione dell'aggiunta.....	3
Il codice dell'aggiunta.....	6
Aggiunte ed eventi.....	9

## Premessa

Nella rubrica "[Visual Basic per Excel](#)" si sono considerate alcune funzioni dell'applicazione di Office manipolabili attraverso la libreria appropriata. Il problema, se non si vuole creare una macro vera e propria, è che spesso risulta scomoda l'interazione tra due applicazioni (quella scritta in Visual Basic e la sessione Excel) separate se si pensa ad esempio che il semplice passaggio da una all'altra sposta il focus sull'applicazione selezionata e quindi rende inutilizzabile l'altra. Fortunatamente Office è stato progettato in modo da poter essere integrato con applicazioni esterne in modo da renderle vere e proprie funzioni interne.

E nel corso di questi capitoli si vedrà nel dettaglio come sfruttare quest' utile caratteristica fornita da Microsoft.

Come già è stato visto per le aggiunte dell'IDE di Visual Basic, è fondamentale tenere a mente i due eventi principali di una qualsiasi applicazione add-in: il collegamento e la chiusura.

Collegare un'aggiunta significa caricarla ed integrarla con l'applicazione a cui si riferisce, scollegarla o chiuderla invece significa eliminarla dalla lista delle aggiunte a disposizione.

E' chiaro che nessuno dei due procedimenti è irreversibile in quanto è possibile collegare e scollegare l'add-in più volte durante un'unica sessione dell'applicazione, a meno che non si sia impostato il collegamento unicamente all'apertura dell'applicazione e lo scollegamento alla chiusura.

Essendo due eventi, collegamento e scollegamento possono essere gestiti in maniera molto semplice dal codice. Proprio come l'evento `Form_Load` si riferisce all'apertura di una form, l'evento `OnConnection` indica il collegamento di un add-in. E come l'evento `Form_Unload` si riferisce alla chiusura di una form, l'evento `OnDisconnection` fa riferimento allo scollegamento di un'aggiunta.

Le modalità di progettazione di un'aggiunta sono:

nessuna: l'aggiunta non sarà caricata all'avvio dell'applicazione e non sarà disponibile all'interno della lista delle aggiunte. Pertanto non potrà mai essere richiamata dall'utente.

A richiesta: l'aggiunta non sarà caricata automaticamente all'avvio dell'applicazione ma sarà elencata nella lista delle aggiunte e perciò a disposizione di un'eventuale richiesta da parte dell'utente.

All'avvio: l'aggiunta sarà collegata automaticamente ad ogni avvio dell'applicazione.

All'avvio successivo: l'aggiunta sarà collegata automaticamente all'avvio successivo dell'applicazione.

## Preparazioni per la creazione di un'aggiunta

Visual Basic prevede un progetto predefinito nel caso si voglia sviluppare un'aggiunta. Per accedere a tale progetto sarà sufficiente aprire un nuovo progetto e nella lista scegliere la voce `AddIn`.

Il codice visualizzato però fa riferimento unicamente alle aggiunte che si integrano con Visual Basic e si rivela pertanto inutile con qualsiasi applicazione di Office (in questo caso Excel).

Basterà quindi cancellare il codice in attesa di svilupparne uno ad hoc.

Si sarà notato che alla creazione del progetto (denominato per default `MyAddIn`) sono state generate due cartelle: `Form` e `Finestre di progettazione`.

La cartella `Form` contiene le form di cui si compone l'aggiunta, ossia la sua interfaccia con la quale l'utente potrà interagire. Ad esempio se si intende creare un'aggiunta che visualizzi in un controllo `ListBox` tutti i valori delle celle selezionate dall'utente, è necessario prevedere una form che contenga la `ListBox`.

La cartella `Finestre di progettazione` contiene invece il modulo `Connect`. Tale modulo è necessario per impostare le modalità di collegamento e scollegamento dell'aggiunta oltre che ad indicare come l'aggiunta si integrerà all'applicazione.

Ad esempio se si vuole inserire un pulsante nella toolbar di Excel, alla cui pressione verrà lanciata l'interfaccia dell'aggiunta, questo dev'essere progettato nel modulo `Connect`.

`Connct` inoltre permette di definire l'applicazione Office alla quale l'aggiunta andrà ad integrarsi.

## La creazione dell'aggiunta

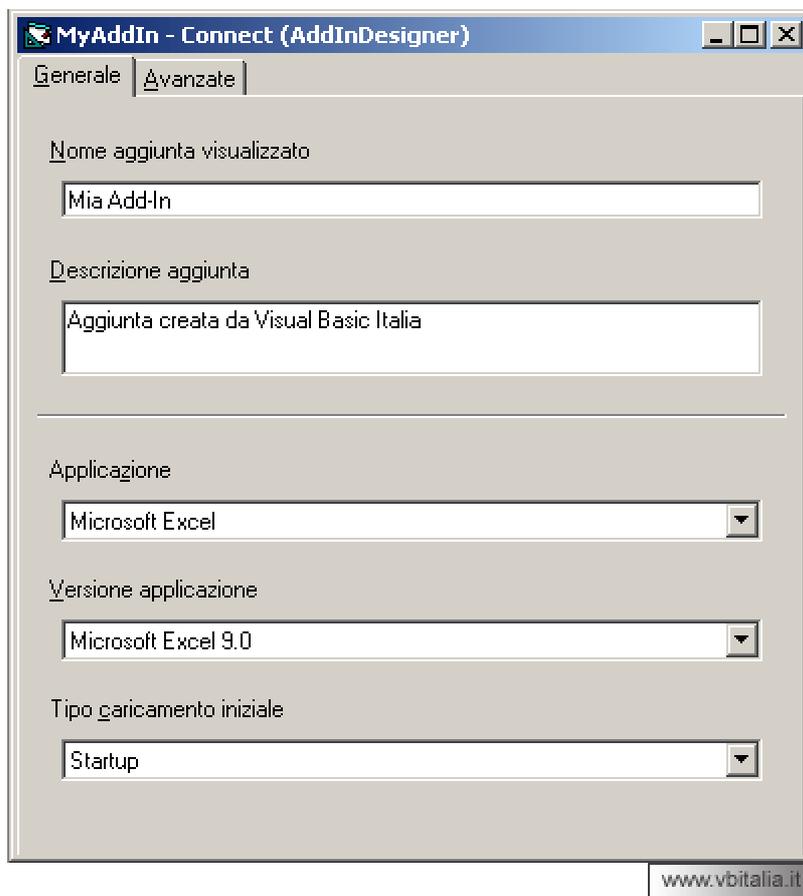
Come già detto la connessione dell'aggiunta permette di avere a disposizione dell'utente l'add-in in caso di successo. Se però si verifica un'errore l'evento restituirà il riferimento all'aggiunta che ha generato l'errore. Sarà semplice in questo modo andare ad isolare e ad eseguire il debug dell'applicazione.

Ora, ancora prima di inserire il codice nel modulo è necessario chiedersi cosa si desidera che faccia l'aggiunta e come si dovrà integrare con l'applicazione Office.

Per rispondere a tali domande basterà visualizzare l'oggetto "modulo Connect". Apparirà in questo modo un pannello sul quale sarà possibile operare varie scelte.

Si è parlato all'inizio del paragrafo di funzioni predefinite, ossia una serie di macro richiamate dall'applicazione Word in corrispondenza di eventi generati dall'utente.

La lista è consultabile da un qualsiasi documento Word scegliendo la voce Strumenti -> Macro dal menu principale e cliccando su Macro (o in alternativa premendo i tasti ALT + F8).



Nella scheda "Generale" è possibile impostare il nome che prenderà l'aggiunta (il nome di default, Mia Add-in va più che bene), una breve descrizione, l'applicazione a cui si integra (ed in questo caso si sceglierà MS Excel) e la modalità di connessione come visto nelle prime righe. Siccome risulta scomodo connettere manualmente l'aggiunta ad ogni avvio di Excel, l'aggiunta si caricherà automaticamente (come indicato dalla voce Startup).

Altra domanda: una volta connessa l'aggiunta, che cosa si desidera che accada ossia, in che modo si vuole che l'aggiunta sia raggiungibile dall'utente? Una soluzione può essere rappresentata dall'immediata visualizzazione dell'interfaccia dell'aggiunta.

Ma quella che si adotterà nell'esempio analizzato inserirà un pulsante nella toolbar di Excel, proprio come mostrato qui sotto.



A tale scopo si dovrà visualizzare il codice del modulo Connect, cancellarne tutto il contenuto come già visto e cominciare ad analizzare gli elementi da aggiungervi.

Il primo passo consiste nello studiare la sintassi dell'evento **OnConnection**:

```
AddinInstance_OnConnection(ByVal Application As Object, ByVal ConnectMode As _
AddInDesignerObjects.ext_ConnectMode, ByVal AddInInst As Object, custom() As
Variant)
```

I parametri dell'evento sono Application, che indica l'applicazione Office alla quale l'aggiunta si integrerà, ConnectMode indica le varie modalità di connessione dell'aggiunta, AddInInst indica invece il riferimento all'aggiunta, utile per richiamare ad esempio il nome dell'aggiunta in ogni parte del codice e custom è un parametro non definito che però può essere utilizzato da chi progetta l'aggiunta.

Dove sono definiti questi parametri? Al momento della connessione (OnConnection), il sistema passa tutti gli elementi elencati sopra alla sottoprocedura. Da quel momento in poi saranno liberamente utilizzabili all'interno del codice.

Ad esempio: una volta connessa, il sistema indica all'aggiunta a quale applicazione si è integrata, (tramite il parametro Application).

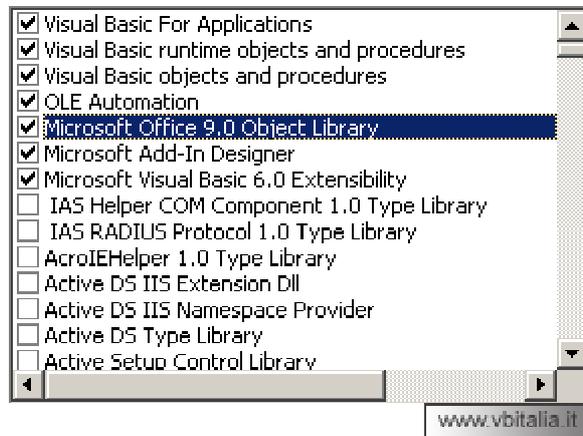
Utilizzando nel resto del codice dell'evento la voce Application, si farà sempre riferimento all'applicazione indicata dal sistema.

La sottoprocedura dell'evento OnConnection sarà dunque inizialmente:

```
Private Sub AddinInstance_OnConnection(ByVal Application As Object, _
ByVal ConnectMode As AddInDesignerObjects.ext_ConnectMode, _
ByVal AddInInst As Object, custom() As Variant)
' ...
' ...
' ...
End Sub
```

Ancora prima di aggiungere codice al posto dei commenti è necessario fare una considerazione di carattere generale ma estremamente importante per la comprensione del progetto: non ci si deve far ingannare dal fatto che l'aggiunta si integri con Excel.

Se infatti nella rubrica "Visual Basic per Excel" la libreria utilizzata era quella di MS Excel, in questo caso si utilizzerà la libreria generale di Office (e per esserne certi basta vedere quali librerie sono state incluse automaticamente all'apertura del progetto AddIn):



Perché?

La ragione specifica sta nel fatto che l'aggiunta in sé stessa non utilizza nessun componente dell'applicazione alla quale si integra: non è necessario definire un'applicazione Excel o una cartella di lavoro Excel, né tantomeno un foglio di calcolo Excel in quanto non saranno utilizzati se non nel momento in cui si andrà a definire i suoi specifici compiti.

Una generica aggiunta infatti, inizialmente senza alcuna funzione per l'utente (ad esempio un'add-in che inserisca un pulsante nella barra di Excel, alla cui pressione mostri una form vuota), utilizzerà unicamente gli elementi generici di una generica applicazione Office come la toolbar, elementi che fanno appunto parte della libreria di Office.

## Il codice dell'aggiunta

Nel corso di questo paragrafo si studierà la struttura di un'aggiunta ad un'applicazione Office, mentre nel prossimo (l'ultimo) verrà visto come far diventare la generica sessione Office, una vera e propria applicazione Excel in modo da poter sfruttare pienamente tutti gli articoli dell'area "Visual Basic per Excel" e permettere all'aggiunta di reagire agli eventi provocati dall'utente sul foglio di calcolo, di lavoro e sull'applicazione Excel.

Fatta la premessa riguardante la differenza tra l'utilizzo della libreria di Office e quella di una qualsiasi applicazione del pacchetto Microsoft, si può ritornare al codice.

Nelle dichiarazioni generali del modulo Connect si andrà quindi a dichiarare un oggetto generico (lo si può denominare appOffice).

Che senso ha tale operazione? Come già visto nel capitolo precedente, una volta collegata l'aggiunta ad Excel, il sistema passa all'evento OnConnection il parametro (Application) che indica quale applicazione è stata integrata.

Per tale motivo se all'interno del codice dell'evento OnConnection si può scrivere:

```
MsgBox Application
```

una finestra di messaggio mostrerà che l'aggiunta è integrata ad Excel ossia che il parametro Application corrisponde a MS Excel.

Ma siccome il parametro è strettamente legato all'evento connessione, risulta pressoché impossibile utilizzarlo in qualsiasi altra sottoprocedura del modulo Connect.

Se esempio nella routine dell'evento OnDisconnection si ripete la stessa linea di codice:

```
MsgBox Application
```

il risultato mostrato dalla finestra di messaggio sarà una stringa vuota "" proprio perché all'interno della routine Application non è definito.

Dichiarando l'oggetto generico appOffice e definendolo (ossia dandogli un valore) all'interno della sottoprocedura OnConnection ci si può riferire all'applicazione Excel in qualunque punto del modulo Connect.

Dunque, il codice finora visto si modifica in questo modo:

```
Dim appOffice As Object
Private Sub AddinInstance_OnConnection(ByVal Application As Object, _
    ByVal ConnectMode As AddInDesignerObjects.ext_ConnectMode, _
    ByVal AddInInst As Object, custom() As Variant)
    On Error Resume Next
    Set appOffice = Application 'applicazione passata dal sistema alla routine
    'attraverso il parametro Application
End Sub
```

Per dimostrare ulteriormente come i parametri vengano passati alla sottoprocedura OnConnection, si può dare un'occhiata anche al parametro AddInInst che indica il riferimento all'aggiunta. Combinando l'oggetto AddInInst e l'oggetto Application, si può programmare l'apertura di una finestra di messaggio alla connessione dell'aggiunta:

```
MsgBox "Caricamento di " & AddInInst & " integrata in " & Application & " _
effettuato."
```

Il risultato sarà del tipo:

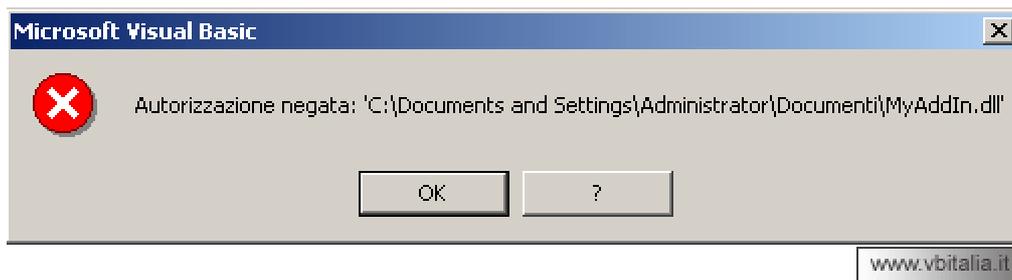


Siccome questo è il primo risultato effettivamente visibile, conviene fare una parentesi: non è sufficiente salvare il progetto perché questo si aggiunga ad Excel.

E' necessario creare un file .dll e salvarlo in una cartella qualsiasi. Il procedimento non necessita di registrazione manuale dell'aggiunta: alla creazione della .dll l'applicazione sviluppata si aggiungerà automaticamente alla lista degli add-in di Excel.

Per ricapitolare, salvare l'intero progetto e successivamente selezionare la voce Crea MyAddIn.dll dalla lista File del menu principale di Visual Basic. Aprendo Excel si potrà quindi visualizzare la finestra di messaggio a cui si accennava poco sopra.

Nota: in alcuni casi può capitare una chiusura non corretta di Excel. In tale eventualità l'aggiunta non verrà scollegata correttamente e pertanto, al momento di un successivo salvataggio della dll si visualizzerà il seguente messaggio di errore:



L'aggiunta infatti risulta ancora connessa ed attiva e pertanto non è possibile modificarla.

Per ovviare al problema chiudere l'applicazione Excel residente in memoria (utilizzando il Task Manager o qualsiasi altro mezzo a disposizione) e salvare nuovamente la dll.

Dopo aver visto le funzioni della variabile oggetto Application è necessario ritornare all'effetto principale dell'aggiunta: l'inserimento di un pulsante nella toolbar di Excel.

Per fare riferimento all'oggetto toolbar di Office è necessario estrarlo dalla libreria di Office, nella quale prende il nome di CommandBar.

Ed un suo singolo elemento, un pulsante generico della toolbar prende il nome di CommandBarButton.

Per questo motivo si può inserire nelle dichiarazioni generali del modulo Connect la dichiarazione di un generico pulsante di una generica toolbar di una generica applicazione Office:

```
Dim WithEvents Pulsante As Office.CommandBarButton
```

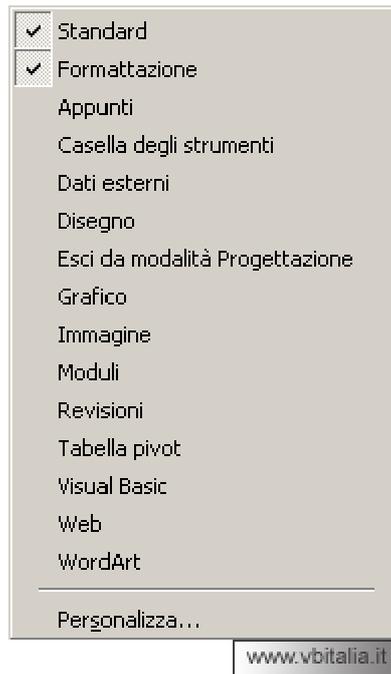
dove WithEvents è una parola chiave che indica alla variabile oggetto Pulsante che dovrà reagire agli eventi generati da un oggetto esterno (in questo caso di CommandBarButton che è esterno in quanto facente parte di della libreria Office, esterna rispetto a Visual Basic).

Una volta dichiarata la variabile oggetto Pulsante, è necessario definirla.

```
Set Pulsante = Application.CommandBars("Standard").Controls.Add(1)
```

Ulteriore parentesi: l'oggetto CommandBar è composto da più elementi quante sono le barre degli strumenti dell'applicazione Office.

Per rendersi conto di questa affermazione basta semplicemente dare una scorsa alla lista tramite la voce "Barre degli strumenti" dal menu "Visualizza":



Indicando la dicitura Standard nella linea di codice appena vista, si indica che variabile oggetto Pulsante sarà un elemento della barra degli strumenti Standard (vedere la figura del pulsante inserito nella barra, pubblicato nel capitolo precedente).

Una volta che è stato definito l'oggetto Pulsante, è possibile assegnargli le proprietà tipiche di un pulsante di una toolbar: il nome, il testo visualizzato al passaggio del mouse e così via.

Iniziando dal nome si può scrivere:

```
With Pulsante
.Caption = "Visual Basic Italia"
```

in secondo luogo si può definire lo stile del pulsante stesso in base ad alcuni valori:

msoButtonAutomatic: adatta il proprio stile a quello del resto dei pulsanti della toolbar;  
 msoButtonIcon imposta il pulsante come icona;  
 msoButtonCaption: imposta il pulsante come solo testo;  
 msoButtonWrapCaption: imposta il pulsante come solo testo adattando le dimensioni alla lunghezza del testo;  
 msoButtonIconAndCaption: imposta il pulsante come testo ed icona affiancati;  
 msoButtonIconAndCaptionBelow: imposta il pulsante come icona con testo allineato in basso;  
 msoButtonIconAndWrapCaption: imposta il pulsante come icona e testo adattando le dimensioni alla lunghezza del testo;  
 msoButtonIconAndWrapCaptionBelow: imposta il pulsante come icona con testo allineato in basso adattando le dimensioni alla lunghezza del testo;

```
.Style = msoButtonCaption
```

In aggiunta si può determinare se il pulsante è abilitato o meno:

```
.Enabled = True ' (False)
```

Nel caso in cui il pulsante sia disabilitato si otterrà un risultato del tipo:



e così via. Le proprietà dell'oggetto CommandBarButton sono così tante che vanno al di là dello scopo di questo singolo articolo.

L'ultima caratteristica che si analizzerà è l'associazione tra pulsante ed aggiunta. Attraverso la proprietà OnAction si determina quale aggiunta (o macro) chiamare in causa nel caso di pressione del pulsante:

```
.OnAction = "!<" & AddInInst.ProgId & ">"  
End With
```

Adesso scorrere la combobox degli oggetti nella finestra di progettazione di Visual Basic. Si vedrà che all'oggetto Pulsante è associata nella ComboBox accanto l'evento Click.

Questo rappresenta chiaramente l'eventualità in cui l'utente clicchi sul pulsante. In tal caso non si farà altro che visualizzare la form di cui si compone il progetto:

```
Private Sub Pulsante_Click(ByVal Ctrl As Office.CommandBarButton, _  
CancelDefault As Boolean)  
frmAddIn.Show  
End Sub
```

L'ultima considerazione dev'essere fatta riguardo alla rimozione dell'aggiunta. In tale eventualità il pulsante dovrà essere eliminato dalla toolbar, mentre l'aggiunta verrà rimossa in modo automatico:

```
Private Sub AddinInstance_OnDisconnection(ByVal RemoveMode As _  
AddInDesignerObjects.ext_DisconnectMode, custom() As Variant) _  
On Error Resume Next  
Pulsante.Delete  
Set Pulsante = Nothing  
Set appOffice = Nothing  
End Sub
```

Col metodo Delete si impone al pulsante di lasciare la propria posizione nella toolbar. Con le due righe di codice successive si svuota la memoria utilizzata fino ad allora per memorizzare la variabile oggetto pulsante e l'applicazione Office.

## Aggiunte ed eventi

In questo paragrafo si vedrà come rendere l'aggiunta sensibile alle modifiche apportate dall'utente sul foglio di calcolo e sull'applicazione Excel.

A tale scopo deve essere fatta una piccola modifica al progetto analizzato nel capitolo precedente: sarà necessario aggiungere un modulo (denominato per impostazione Module1) e lì dichiarare come globale la variabile oggetto appOffice.

```
Global appOffice As Object
```

Questo perchè adesso si utilizzerà anche Form1 oltre al modulo Connect: la variabile che indica l'applicazione Office deve perciò essere valida in entrambi i moduli di codice.

All'interno del modulo Module1 è inoltre possibile inserire il codice che permette all'aggiunta di essere sempre in primo piano rispetto al foglio di calcolo, per evitare che venga nascosta una volta attivato il foglio. Come mostrato nel codice presente nell'Archivio quindi:

```
Sub AlwaysOnTop(FrmID As Form, OnTop As Integer)  
Const SWP_NOMOVE = 2  
Const SWP_NOSIZE = 1  
Const FLAGS = SWP_NOMOVE Or SWP_NOSIZE  
Const HWND_TOPMOST = -1  
Const HWND_NOTOPMOST = -2  
If OnTop = -1 Then  
OnTop = SetWindowPos(FrmID.hwnd, HWND_TOPMOST, 0, 0, 0, 0, FLAGS)  
Else
```

```

OnTop = SetWindowPos(FrmID.hwnd, HWND_NOTOPMOST, 0, 0, 0, 0, FLAGS)
End If
End Sub

```

Adesso si può passare al codice di Form1. Siccome si utilizzeranno gli oggetti propri di una sessione Excel, è necessario importare la libreria relativa all'interno dell'applicazione, come già visto negli articoli della rubrica 'Visual Basic per Excel'.

Si provi a scrivere una cosa del genere:

```

Dim sessioneExcel As Excel.Application
Dim cartExcel As Excel.Workbook
Dim foglioExcel As Excel.Worksheet

```

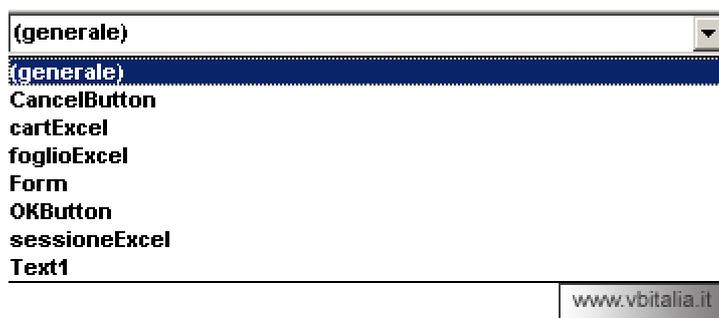
E si controlli nella combobox della finestra di Visual Basic quali oggetti sono a disposizione. Adesso si modifichi il codice appena visto in questo modo:

```

Dim WithEvents sessioneExcel As Excel.Application
Dim WithEvents cartExcel As Excel.Workbook
Dim WithEvents foglioExcel As Excel.Worksheet

```

E si controlli nuovamente la combobox: sono presenti tre nuovi oggetti coi relativi eventi: cartExcel, foglioExcel e sessioneExcel.



Questo perché WithEvents rende l'applicazione sensibile alle modifiche apportate sugli oggetti caratteristici di Excel.

In questo modo ad esempio si può visualizzare una finestra di messaggio ogni volta che l'utente seleziona una cella specifica e più in generale si possono utilizzare tutti gli eventi di cui si compone la libreria di Excel.

Adesso si consideri la sottoprocedura Load di Form1:

```

Private Sub Form1_Load()
' ...
' ...
' ...
End Sub

```

Il problema ulteriore è di più difficile comprensione: nel modulo Connect è stata dichiarata e definita una variabile oggetto indicante una generica applicazione Office: appOffice.

Nel modulo di codice di Form1 invece si utilizza una seconda applicazione, sessioneExcel. Come indicare all'aggiunta che queste due applicazioni sono in realtà la stessa?

Semplicemente eguagliando le due variabili oggetto, nel seguente modo:

```

Set sessioneExcel = appOffice

```

Se infatti si fosse dichiarata sessioneExcel come in tutti gli altri articoli in questo modo:

```

Dim sessioneExcel As New Excel.application

```

si sarebbe creata una nuova sessione Excel, e ci si sarebbe quindi trovati con due applicazioni Excel: appOffice e sessioneExcel. In quel caso non si sarebbe potuto utilizzare né il foglio di calcolo dell'applicazione Excel in cui è stata caricata l'aggiunta, né la cartella di lavoro.

Altro problema: appOffice, nella quale è caricata l'aggiunta, apre automaticamente un nuovo foglio di calcolo all'interno di una nuova cartella di lavoro.

Com'è possibile indicare all'aggiunta di operare proprio su quel foglio e su quella cartella, invece di aprirne di nuovi? Attraverso gli oggetti ActiveSheet ed ActiveWorkbook:

```
Set cartExcel = sessioneExcel.ActiveWorkbook
Set foglioExcel = sessioneExcel.ActiveSheet
```

che rappresentano rispettivamente la cartella di lavoro ed il foglio di calcolo correntemente utilizzati dall'applicazione Excel.

Per chiudere la sottoprocedura, si può inserire la linea di codice che, richiamando la routine di Module1, consenta di visualizzare la finestra dell'aggiunta in primo piano rispetto al foglio di calcolo:

```
AlwaysOnTop Me, -1
```

Adesso il più del lavoro è terminato. Quello che rimane consiste solamente nell'applicazione dei concetti visti nel corso degli articoli della rubrica 'Visual Basic per Excel', riferiti questa volta ai vari eventi degli oggetti sessione Excel, cartella di lavoro Excel e foglio di calcolo Excel.

Un semplice esempio può essere rappresentato proprio da quello che era già stato anticipato: un controllo TextBox su Form1 che visualizzi il valore della cella selezionata dall'utente.

Parlando di celle si fa dunque riferimento all'oggetto foglioExcel e parlando di selezione ci si riferisce all'evento SelectionChange dell'oggetto Excel.Workbook:

```
Private Sub foglioExcel_SelectionChange(ByVal Target As Excel.Range)
' ...
' ...
' ...
End Sub
```

dove al posto dei commenti si può inserire l'assegnazione del valore della cella alla casella di testo:

```
On Error Resume Next
Text1.Text = Target.Value
```

E' però necessario notare due cose, legate tra loro: il parametro Target dell'evento è (come indica la definizione Excel.Range) un Range, ossia un insieme di celle.

E' impossibile visualizzare pertanto il valore delle celle nella casella di testo se queste sono più di una.

Ecco la ragione della linea di codice precedente (On Error Resume Next) che serve proprio ad evitare l'errore in questi casi.

Naturalmente l'aggiunta può essere ben più complessa rispetto a questo semplice esempio ma il suo scopo è quello di rendere l'idea di come adesso sia possibile utilizzare non soltanto gli oggetti e le relative proprietà della libreria di Office, ma anche tutti gli eventi da essi supportati.

Ed il risultato di un'applicazione di questo genere può essere rappresentato graficamente dall'immagine sottostante:

