

**CISA1**  
**Prima Conferenza Italiana Sviluppatori Access**  
**Arezzo, 4-5 giugno 2005**

**WithEvents e RaiseEvent**

**Gestione degli eventi predefiniti e creazione di nuovi eventi  
in Access VBA**

Marco Pizzamiglio  
namor@inwind.it  
Gradisca d'Isonzo (GO) - ITALY

## **Indice**

<b>1. WITHEVENTS - CHE COS'E'</b> .....	<b>3</b>
USI PRATICI: SI PUÒ USARE PER .....	3
<b>2. WITHEVENTS - COME SI USA</b> .....	<b>3</b>
DICHIAZIONE DELLA VARIABILE .....	3
ASSOCIARE LA VARIABILE ALL'OGGETTO ED ABILITARE L'EVENTO .....	4
SCRIVERE IL CODICE PER L'EVENTO .....	4
IL CODICE DELLA CLASSE .....	5
CREARE UN'ISTANZA DELLA CLASSE.....	6
<b>4. RIEPILOGANDO</b> .....	<b>7</b>
<b>5. NOTE</b> .....	<b>7</b>
<b>6. UN PROBLEMA PRATICO: GESTIRE UN GRUPPO DI CONTROLLI</b> .....	<b>8</b>
ANALISI DEL PROBLEMA.....	8
IL MODULO DI CLASSE .....	9
UNA PRIMA SOLUZIONE .....	10
OTTIMIZZAZIONE .....	10
CICLO SULL'INSIEME CONTROLS .....	11
SALVARE LE ISTANZE DELLA CLASSE .....	11
IL CODICE DELLA MASCHERA .....	12
<b>7. SISTEMA DI LOG AUTOMATICO PER MASCHERE</b> .....	<b>13</b>
ANALISI DEL PROBLEMA.....	13
REALIZZAZIONE .....	13
OTTIMIZZAZIONE .....	14
<b>8. RAISEEVENT - CHE COS'E'</b> .....	<b>16</b>
<b>9. RAISEEVENT - COME SI USA</b> .....	<b>16</b>
<b>10. UN ESEMPIO PRATICO: AVVISO IN FASE DI LOG</b> .....	<b>17</b>
<b>11. QUANDO E PERCHÉ USARE RAISEEVENT</b> .....	<b>20</b>

## **1. WithEvents - CHE COS'E'**

E' una keyword di vba che consente ad una variabile oggetto di intercettare gli eventi generati da un modulo di classe.

L'help in linea di Access ne dice molto poco e non riporta esempi.

In certe situazioni consente di risolvere con poche righe di codice problemi altrimenti irrisolvibili o che richiederebbero pesanti lavori di copia-incolla.

### **USI PRATICI: si può usare per**

- semplificare la gestione degli eventi di un gruppo di controlli dello stesso tipo (per esempio un gruppo di Textbox, Combobox, Checkbox, Forms, Reports, ecc...)
- gestire gli eventi generati da un oggetto d'automazione (Word, Excel, ecc... o altri moduli di classe)

## **2. WithEvents - COME SI USA**

### **Dichiarazione della variabile**

Può essere usato solo in un modulo di classe, pertanto dovremo creare nel database un nuovo modulo di classe (Inserisci/Modulo di classe).

Si aggiunge alla riga di dichiarazione di una variabile, per esempio:

```
Private WithEvents Casella as TextBox
```

Bisogna sempre specificare il tipo di oggetto di cui vogliamo gestire gli eventi, non può essere usato con dichiarazioni generiche come:

```
Private WithEvents Casella as Control <--- NO!
```

## Associare la variabile all'oggetto ed abilitare l'evento

Una volta dichiarata la variabile dobbiamo:

- associarla all'oggetto di cui vogliamo gestire gli eventi:

```
Set Casella = Forms("Anagrafica").Cognome
```

- abilitare la gestione dell'evento che vogliamo gestire

```
Casella.AfterUpdate="[Event Procedure]"
```

Questa forma è poco flessibile perché consente di gestire solo un oggetto, la casella Cognome della maschera Anagrafica.

Conviene creare una Function a cui passeremo come parametro l'oggetto al quale associare la variabile:

```
Public Function associa(Oggetto_da_gestire As TextBox)
```

```
Set Casella = Oggetto_da_gestire
```

```
Casella.AfterUpdate="[Event Procedure]"
```

```
End Function
```

La routine dev'essere pubblica perché dovrà essere richiamata dall'esterno del modulo, e trattandosi di un modulo di classe diventerà un Metodo della classe.

Per abilitare l'evento, anziché la forma italiana "[Routine Evento]" conviene usare la forma inglese "[Event Procedure]", che dovrebbe essere riconosciuta anche dalle versioni di Access in altre lingue.

## Scrivere il codice per l'evento

Dopo aver dichiarato la variabile e averla associata all'oggetto, dobbiamo scrivere la/le Sub per gli eventi che vogliamo gestire:

```
Private Sub Casella_AfterUpdate()
```

```
Msgbox "Hai modificato la casella " & Casella.Name
```

```
End Sub
```

Da notare che il nome dell'evento è lo stesso che useremo nel codice di una maschera o report, ma al posto del nome dell'oggetto usiamo la variabile dichiarata con WithEvents.

All'interno del codice possiamo usare tutti i metodi e le proprietà dell'oggetto che stiamo gestendo.

Per esempio, se pensiamo di usare questa classe per gestire più di una textbox, con la proprietà *Casella.Name* possiamo sapere qual è quella da cui è stato generato l'evento.

## Il codice della classe

A questo punto la nostra classe contiene tre cose:

- la dichiarazione della variabile con WithEvents
- il metodo (*Public Sub associa*) che useremo per associare la variabile alla textbox da gestire e abilitare l'evento
- la routine evento (*Casella\_AfterUpdate*) contenente il codice che verrà eseguito ogni volta che modificheremo il valore contenuto nella textbox

```
Private WithEvents Casella as TextBox
```

```
Public Sub associa(Oggetto_da_gestire As TextBox)  
    Set Casella = Oggetto_da_gestire  
    Casella.AfterUpdate="[Event Procedure]"  
End Sub
```

```
Private Sub Casella_AfterUpdate()  
    Msgbox "Hai modificato la casella " & Casella.Name  
End Sub
```

Il compito che svolge è mostrare un avviso ogni volta che modifichiamo il valore contenuto nella textbox che stiamo gestendo. Salviamo il modulo di classe assegnandogli un nome, per esempio "Textbox\_Avviso\_di\_modifica".

## Creare un'istanza della classe

Supponiamo di voler abilitare l'avviso per le due textbox "Cognome" e "Data nascita" su una maschera Anagrafica.

All'interno della maschera dovremo dichiarare due variabili di tipo *Textbox\_Avviso\_di\_modifica* ed associarle alle due caselle da gestire:

```
Private AvvisaCognome As Textbox_Avviso_di_modifica  
Private AvvisaDataNascita As Textbox_Avviso_di_modifica
```

```
Private Sub Form_Load()  
    Set AvvisaCognome = New Textbox_Avviso_di_modifica  
    AvvisaCognome.associa Me.Cognome  
    Set AvvisaDataNascita = New Textbox_Avviso_di_modifica  
    AvvisaDataNascita.associa Me.Data_nascita  
End Sub
```

Proviamo ad aprire la maschera e ad inserire un valore in una delle due caselle, dovrebbe comparire l'avviso.

## **4. RIEPILOGANDO**

In un modulo di classe dovremo avere tre cose:

- la variabile oggetto dichiarata con WithEvents
- il metodo (*Public Sub*) che useremo per associare la variabile all'oggetto da gestire ed abilitare l'auditing degli eventi
- la/le sub contenenti il codice da eseguire al verificarsi degli eventi

Nel codice della maschera dovremo:

- dichiarare e creare una nuova istanza della classe per ciascun controllo da gestire
- associare la classe all'oggetto da gestire, usando l'apposito metodo

## **5. NOTE**

Se l'oggetto ha già del codice vba su uno dei suoi eventi esso verrà eseguito prima del codice gestito con WithEvents.

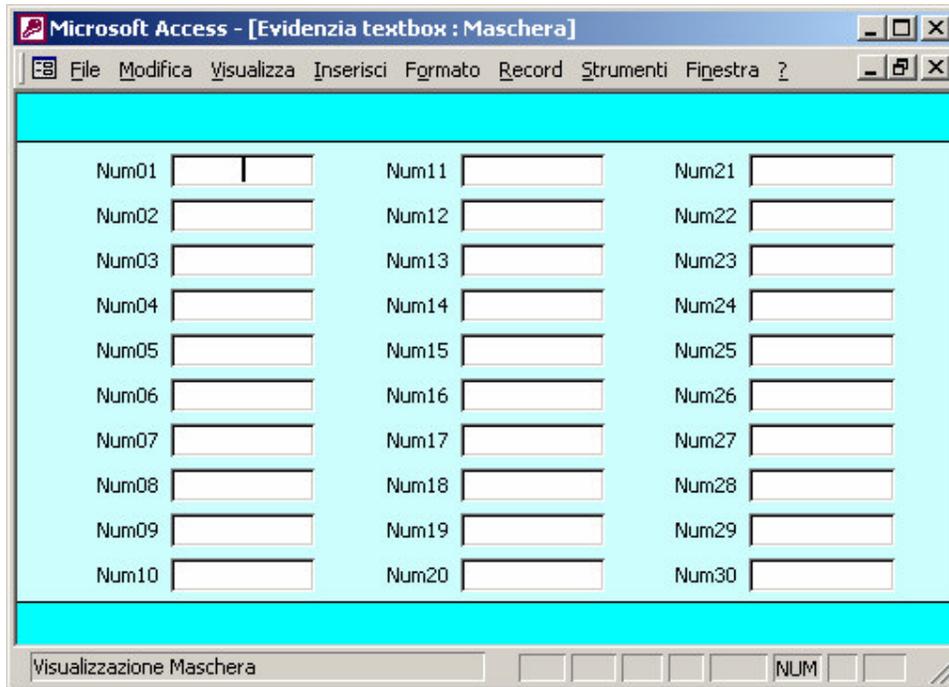
Supponiamo di avere una maschera su cui c'è una textbox 'Cognome' che ha sia un suo evento *Private Sub Cognome\_Click* che un evento gestito con WithEvents (*Sub Casella\_Click*).

Al verificarsi dell'evento verranno eseguite entrambe, prima la *Sub Cognome\_Click* e poi la *Sub Casella\_Click*.

La realizzazione pratica di quanto fin qui spiegato si trova in formato A97 e A00 nei due database "Axx Esempio 1 WithEvents.mdb" allegati.

## **6. UN PROBLEMA PRATICO: Gestire un gruppo di controlli**

Supponiamo di avere una maschera con trenta textbox:



Vogliamo evidenziare con grassetto e fondo giallo la casella in cui si trova il cursore. In più vogliamo impedire che in ciascuna casella vengano inseriti più di tre caratteri.

### **Analisi del problema**

Dovremo gestire tre eventi per ciascuna textbox:

- su OnEnter (quando il cursore entra nella textbox) coloriamo il fondo di giallo ed applichiamo il grassetto
- su OnExit (quando il cursore esce dalla textbox) togliamo il grassetto e rimettiamo il fondo bianco
- su BeforeUpdate verificiamo la lunghezza del testo inserito, se è superiore ai tre caratteri visualizziamo un avviso e blocchiamo l'inserimento impostando Cancel=True

Con il metodo classico dovremmo scrivere tre sub evento per trenta caselle = NOVANTA sub!

## Il modulo di classe

Questo il codice del modulo di classe che gestisce i tre eventi:

```
Private WithEvents Casella As TextBox
```

```
Private Sub Casella_BeforeUpdate(Cancel As Integer)
```

```
    If Len(Casella) > 3 Then  
        MsgBox "Non puoi inserire più di 3 caratteri nella casella '" &  
Casella.Name  
        Cancel = True  
    End If  
End Sub
```

```
Private Sub Casella_Enter()
```

```
    Casella.BackColor = vbYellow  
    Casella.FontBold = True  
End Sub
```

```
Private Sub Casella_Exit(Cancel As Integer)
```

```
    Casella.BackColor = vbWhite  
    Casella.FontBold = False  
End Sub
```

```
Public Function associa(TB As TextBox)
```

```
    Set Casella = TB  
    Casella.BeforeUpdate = "[Event Procedure]"  
    Casella.OnEnter = "[Event Procedure]"  
    Casella.OnExit = "[Event Procedure]"  
End Function
```

## Una prima soluzione

Seguendo la logica dell'esempio precedente dovremmo dichiarare trenta variabili per trenta diverse istanze della classe, che chiameremo *Evidenzia\_textbox*, ed associare una per una le classi ai controlli:

```
Private Casella01 As Evidenzia_textbox
Private Casella02 As Evidenzia_textbox
.....
.....
Private Casella30 As Evidenzia_textbox

Set Casella01 = New Evidenzia_textbox
Casella01.associa Me.TextBox01
Set Casella02 = New Evidenzia_textbox
Casella02.associa Me.TextBox02
.....
.....
Set Casella30 = New Evidenzia_textbox
Casella30.associa Me.TextBox30
```

## Ottimizzazione

Per semplificare la creazione e l'associazione delle classi possiamo usare un ciclo. Siccome non tutti i controlli sulla maschera dovranno avere associata un'istanza della classe (sulle etichette non serve) bisogna scegliere un sistema per individuare quelli da gestire. Un ottimo e veloce metodo è l'inserimento di un flag nella proprietà Tag dei controlli. Facciamo una selezione multipla su tutte le textbox da gestire e scriviamo una X nella proprietà Tag.

## Ciclo sull'insieme Controls

A questo punto un semplice ciclo sull'insieme Controls della maschera ci permette di individuare le textbox a cui associare la classe *Evidenzia\_textbox*:

```
Private Sub Form_Load()  
Dim EvidTxtb As Evidenzia_textbox, Ctl As Control  
For Each Ctl In Me.Controls  
If Ctl.Tag = "X" Then  
Set EvidTxtb = New Evidenzia_textbox  
EvidTxtb.associa Ctl  
End If  
Next  
End Sub
```

Questo codice non può funzionare per due motivi:

- avendo un'unica variabile di tipo *Evidenzia\_textbox* essa viene riscritta ad ogni iter del ciclo For, quindi rimarrebbe valida solamente per l'ultima delle textbox
- essendo stata dichiarata all'interno della *Sub Form\_Load* la variabile *EvidTxtb* (e quindi la classe che gestisce gli eventi) cessa di esistere alla fine della Sub, quindi nemmeno l'ultima delle textbox sarà gestita dalla classe

## Salvare le istanze della classe

Per far sì che le trenta istanze della classe *Evidenzia\_textbox* rimangano attive bisognerà salvarle da qualche parte. Conoscendo il numero di textbox da gestire potremmo usare un array dichiarato a livello di maschera in cui memorizzare ciascuna istanza di *Evidenzia\_textbox* creata:

```
Private Istanze(1 To 30) As Evidenzia_textbox
```

Altro sistema, più flessibile in quanto non abbiamo bisogno di conoscere a priori il numero di textbox, è l'uso di una *Collection*:

```
Private Istanze As New Collection
```

sempre dichiarata a livello di maschera, di modo che rimanga attiva finché la maschera è aperta, e al momento della chiusura della maschera venga distrutta liberando la memoria occupata.

## Il codice della maschera

*Private Istanze As New Collection*

*Private Sub Form\_Load()*

*Dim EvidTxtb As Evidenzia\_textbox, Ctl As Control*

*For Each Ctl In Me.Controls*

*If Ctl.Tag = "X" Then*

*Set EvidTxtb = New Evidenzia\_textbox*

*EvidTxtb.associa Ctl*

*Istanze.Add EvidTxtb*

*End If*

*Next*

*End Sub*

Come si può notare, per ogni controllo sulla maschera che abbia una 'X' nella proprietà *Tag* viene:

- crea una nuova istanza della classe *Evidenzia\_textbox*
- richiamato il metodo *associa* che la lega alla textbox
- salvata l'istanza nella *Collection*

Alla fine del ciclo *For* la *Collection Istanze* conterrà trenta classi *Evidenzia\_textbox* associate alle altrettante textbox sulla maschera. Alla chiusura della maschera la *Collection* verrà distrutta assieme a tutte le trenta istanze della classe.

La realizzazione pratica di questo esempio si trova in formato A97 e A00 nei due database "Axx Esempio 2 WithEvents.mdb" allegati.

## **7. Sistema di log automatico per maschere**

L'obiettivo è la realizzazione di un sistema di log automatico ed universale, applicabile a qualsiasi maschera associata, che registri in un file di testo le operazioni eseguite sui dati tramite la maschera.

Registreremo il nome dell'utente che ha modificato i dati, il nome della maschera da cui proviene il log, data e ora della modifica, i valori (prima e dopo la modifica) di tutti i campi dei record modificati, i valori di tutti i campi dei nuovi record inseriti e dei record eliminati.

### **Analisi del problema**

Dovremo gestire parecchi eventi, e salvare (con un ciclo sul RecordsetClone della maschera) il valore di tutti i campi in una variabile che verrà scritta sul file:

Su Eliminazione, Prima e Dopo conferma eliminazione, Prima e dopo Aggiornamento, Prima e Dopo Inserimento

Per far sì che il sistema di log sia applicabile a qualsiasi maschera con minimo sforzo dovremo scrivere tutto il codice degli eventi in un modulo di classe usando WithEvents, in modo che si possa poi creare un'istanza della classe da associare alla Form da monitorare.

### **Realizzazione**

L'uso di WithEvents per gestire gli eventi delle maschere segue la stessa logica:

un modulo di classe (*TextLOG*) con:

- la variabile oggetto (di tipo *Form*) dichiarata con WithEvents
- il metodo (*Public Sub*) che useremo per associare la variabile alla Form da gestire ed abilitare l'auditing degli eventi sulla Form
- le sub contenenti il codice da eseguire al verificarsi degli eventi della Form

(data la sua lunghezza il listato del modulo *TextLOG* non è riportato qui, ma è reperibile nell'esempio allegato ed è ampiamente commentato)

Nel codice della maschera dovremo:

- dichiarare e creare una nuova istanza della classe
- associare la classe alla Form da gestire

```
Private FormTextLog As TextLOG
```

```
Private Sub Form_Open(Cancel As Integer)  
    Set FormTextLog = New TextLOG  
    FormTextLog.Associa Form  
End Sub
```

## Ottimizzazione

In questo caso non ci sono problemi di istanze multiple in quanto ogni Form ha al suo interno una sola variabile *Private FormTextLog As TextLOG* che istanzia la classe, e che viene automaticamente distrutta alla chiusura della form.

E però possibile ottimizzare ulteriormente il sistema di log in modo che non ci sia bisogno di scrivere nemmeno le tre righe di codice appena viste.

Sarà necessario creare un modulo che contenga

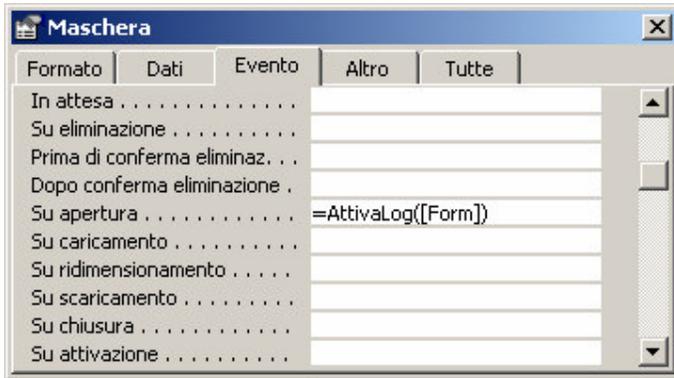
- una *Public Collection* dove salvare le istanze della classe *TextLOG*, una per ogni maschera aperta
- una routine che riceva come parametro un riferimento alla Form su cui abilitare il log, e che si occupi di creare una nuova istanza della classe, associarla alla Form e salvarla nella *Public Collection*:  

```
Public Function AttivaLog(Maschera As Form)
```
- una *Public Sub* che, alla chiusura di una Form, si occupi di rimuovere dalla *Collection* la relativa istanza, altrimenti rimarrebbe attiva sprestando memoria.

Il sistema di log si potrà attivare con un'unica riga di codice richiamando la routine *AttivaLog*, passandole come parametro un riferimento alla Form da monitorare:

```
AttivaLog([Form])
```

Trattandosi di un'unica istruzione è possibile scriverla direttamente nella finestra delle proprietà, nella riga dell'evento *Apertura* o *Caricamento* della Form.



Perché il sistema funzioni è indispensabile che la proprietà 'Possiede modulo' della Form sia settata a Sì, altrimenti il codice della classe non verrà eseguito.

La realizzazione pratica di questo esempio si trova in formato A97 e A00 nei due database "Axx Esempio 3 WithEvents.mdb" allegati.

## **8. RaiseEvent - CHE COS'E'**

E' una keyword di vba che, utilizzata in coppia con la keyword *Event*, consente ad un modulo di classe di generare un evento personalizzato. L'evento così generato potrà essere intercettato e gestito dal codice di un altro modulo di classe, per esempio da una maschera o da un report, utilizzando *WithEvents*.

## **9. RaiseEvent - COME SI USA**

L'uso di *RaiseEvent* è molto semplice. All'interno del modulo di classe si dichiarano a livello di modulo (quindi fuori da tutte le sub) gli eventi che vogliamo siano generati:

*Event NomeEvento()*

e all'interno del codice del modulo, al verificarsi delle condizioni che devono generare l'evento:

*If condizione=True Then RaiseEvent NomeEvento*

NOTA: *Event* e *RaiseEvent* sono disponibili a partire da Access 2000.

## **10. UN ESEMPIO PRATICO: Avviso in fase di log**

Supponiamo di avere in un database una maschera associata ad una tabella anagrafica, e un modulo di classe *LOG\_su\_file* progettato per registrare nel file di testo "C:\LOGfile.txt" delle stringhe che gli vengono passate. All'interno della maschera creiamo un'istanza della classe *LOG\_su\_file* per tenere traccia di quali record sono stati visti, e ogni volta che un record viene visualizzato sulla maschera (su evento *Corrente*) passeremo alla classe il nome affinché venga scritto sul file assieme alla data e l'ora.

Questo è il codice del modulo di classe *LOG\_su\_file*:

```
Private Riga As String
Private NumFile As Integer

Private Sub Class_Initialize()
    NumFile = FreeFile
    Open "C:\LOGfile.txt" For Output As #NumFile
    Print #NumFile, "CISA1 - Arezzo 4-5 giugno 2005"
End Sub

Private Sub Class_Terminate()
    Close NumFile
End Sub

Public Sub Scrivi(Testo As String)
    Riga = Now & " - " & Testo
    Print #NumFile, Riga
End Sub
```

E questo è il codice della maschera:

```
Private LOG As New LOG_su_file

Private Sub Form_Current()
    LOG.Scrivi (Me.Nome)
End Sub
```

Ora facciamo in modo che il modulo di classe tenga il conto del numero di caratteri che scrive sul file, e generi un evento non appena superiamo i 100 caratteri.

Dobbiamo aggiungere una variabile *NumChar* per conteggiare i caratteri, dichiarare l'evento *TroppiCaratteri* e generarlo con *RaiseEvent* appena superiamo i 100:

```
Private Riga As String
Private NumFile As Integer
Private NumChar As Long 'numero dei caratteri scritti sul file

Event TroppiCaratteri()

Private Sub Class_Initialize()
    NumFile = FreeFile
    Open "C:\LOGfile.txt" For Output As #NumFile
    Print #NumFile, "CISA1 - Arezzo 4-5 giugno 2005 - Esempio RaiseEvent"
    NumChar = 0
End Sub

Private Sub Class_Terminate()
    Close NumFile
End Sub

Public Sub Scrivi(Testo As String)
    Riga = Now & " - " & Testo
    Print #NumFile, Riga
    NumChar = NumChar + Len(Riga)
    If NumChar > 100 Then RaiseEvent TroppiCaratteri
End Sub
```

Il modulo della maschera andrà modificato per poter gestire il nuovo evento:

```
Private WithEvents LOG As LOG_su_file
```

```
Private Sub Form_Open(Cancel As Integer)
```

```
    Set LOG = New LOG_su_file
```

```
End Sub
```

```
Private Sub Form_Current()
```

```
    LOG.Scrivi (Me.Nome)
```

```
End Sub
```

```
Private Sub LOG_TroppiCaratteri()
```

```
    MsgBox "Hai scritto sul file più di 100 caratteri!", , "Esempio RaiseEvent"
```

```
End Sub
```

NOTA: Finché la maschera rimane aperta il file di testo appare vuoto, perché le righe che scriviamo sul file rimangono in un buffer di transito fino a quando viene eseguita l'istruzione *Close* che ne forza la scrittura su disco. L'istruzione *Close* viene eseguita quando il modulo di classe termina, cioè quando viene chiusa la maschera. Dovremo pertanto chiudere la maschera per poter esaminare il contenuto del file.

La realizzazione pratica di questo esempio si trova in formato A00 nel database "A00 Esempio 4 RaiseEvent.mdb" allegato.

## **11. Quando e perché usare *RaiseEvent***

Avremmo ottenuto lo stesso risultato mettendo il *Msgbox* di avviso direttamente all'interno del modulo di classe al posto del *RaiseEvent*, o mettendolo in una *Public Function* di un altro modulo in modo da poterlo richiamare anche da diversi punti della classe *LOG\_su\_file*.

Se però volessimo ottenere su due maschere due comportamenti diversi, un semplice avviso sulla prima e un blocco di sicurezza sulla seconda in modo da impedire di loggare più di un certo numero di caratteri, dovremmo usare due moduli identici in tutto tranne per il fatto che in uno avremo l'*Msgbox* di avviso e nell'altro un'istruzione per bloccare o chiudere la maschera.

La soluzione con *RaiseEvent* sarebbe la più indicata perché potremmo usare lo stesso modulo di classe ed implementare i due diversi comportamenti all'interno di routine evento nel codice delle due maschere.

Usiamo *RaiseEvent* anche quando stiamo sviluppando un modulo da distribuire ad altri sviluppatori e per qualche motivo non vogliamo rilasciare il codice sorgente, o non vogliamo costringerli ad andarsi a studiare tutto il codice per scoprire dove e come modificarlo per ottenere gli effetti desiderati. Mettendo a disposizione alcuni eventi sarà facile per chi usa il nostro modulo personalizzarne il comportamento senza aver bisogno di leggersi il codice, che in alcuni casi può essere piuttosto lungo e complesso.

Basti pensare al funzionamento di Access: pur senza avere accesso al codice sorgente di maschere, report e controlli, siamo in grado di personalizzarne il comportamento sfruttando gli eventi messi a disposizione dai programmatori. Se dovessimo ottenere gli stessi risultati senza la gestione degli eventi dovremmo avere a disposizione il codice sorgente di Access ed andarci a leggere decine di migliaia di righe di codice per trovare il punto in cui una maschera carica un record (evento *Current*), o dove un record viene modificato (eventi *BeforeUpdate* e *AfterUpdate*), ecc... per potervi aggiungere il nostro codice personalizzato.

*RaiseEvent* è dunque un ottimo sistema per creare librerie di funzioni specifiche il cui comportamento può essere di volta in volta esteso e personalizzato sfruttando la gestione degli eventi.