Gestione degli errori - Metodi e proprietà

La *gestione degli errori* si riferisce ad una particolare tecnica di programmazione che consiste nell'anticipare e prevedere le condizioni di errore che possono sorgere quando il programma viene eseguito. In generale gli errori possono essere di tre tipi:

- Errori di sintassi: Come errori di battitura o variabili non dichiarate che impediscono il corretto funzionamento
- Errori run-time: Che si verificano quando VBA non può eseguire correttamente una dichiarazione del programma.
- Errori Logici: Come un utente immette un valore negativo in cui solo un numero positivo è accettabile

Errori di sintassi

Viene detta sintassi lo specifico ordine di parole che costituisco un enunciato VBA valido e alcuni dei più comuni messaggi di errore che possono verificarsi mentre si scrive o si edita una procedura sono relativi a errori di sintassi. I messaggi di errore di sintassi segnalano un problema in un enunciato quali, virgole, virgolette argomenti mancanti o simili. Ogni volta che si scrive una nuova riga di codice o se ne modifica una, il VBA esamina la riga non appena il cursore si sposta da essa. Questa operazione che viene detta ***Parsing*** è il processo di scomposizione di un enunciato nelle varie parti al fine di determinare quali sono le parole chiave, le variabili o i dati. Dopo che il VBA ha esaminato senza rilevare errori una riga di codice, procede alla sua compilazione (in VBA la compilazione è la conversione del codice sorgente in una forma direttamente eseguibile dal VBA senza dover ripetere l'operazione di Parsing)

Si deve tenere presente che la finestra del codice nell'Editor di VB presenta le parole scritte in vari colori, questa "applicazione" del colore viene eseguita dopo aver esaminato una riga di codice e non aver rilevato errori, al contrario, se viene rilevato un errore in fase di esame o compilazione della riga il VBA colora di rosso l'intera riga e visualizza una finestra di dialogo con un messaggio di errore

Errori Run-time

E' possibile scrivere un enunciato VBA senza alcun errore di sintassi, ma che tuttavia non porta a un'esecuzione corretta, gli errori che si manifestano solo in fase di esecuzione della procedura vengono detti errori runtime. Ne esistono di vari tipi e di solito sono causati dalla mancanza di argomenti o dall'uso di argomenti di tipo errato, dalla mancanza di certe parole chiave o dal tentativo di accedere a dischi o directory non esistenti o da errori logici.

Errori logici

Questo tipo di errore è il più difficile da tracciare se la sua sintassi è corretta e funziona senza rimandare errori di esecuzione, si può definire come un errore da nessuna indicazione all'utente che si è verificato un errore dovuto al fatto che un errore logico è il processo di logica e non il codice stesso a generare l'errore. L'esecuzione di un calcolo in un foglio di lavoro utilizzando una funzione restituirà una risposta, ma è la risposta corretta? Per meglio comprendere simuliamo un errore scrivendo il codice sotto riportato in un modulo e mandiamolo in esecuzione

```
Sub prova()
Sheets("Foglio11").Range("F5").Select
End Sub
```

Il cui significato è: *vai nel Foglio 11* di questa cartella e seleziona (*Select*) la cella F5 *Range("F5")*. Tornando ad Excel (basta cliccare sulla prima icona col simbolo di Excel in alto a sinistra) e mandando in esecuzione la macro (premere **ALT+F8**, selezionare il nome della macro e poi *Esegui*) e comparirà un avviso come quello sotto riportato

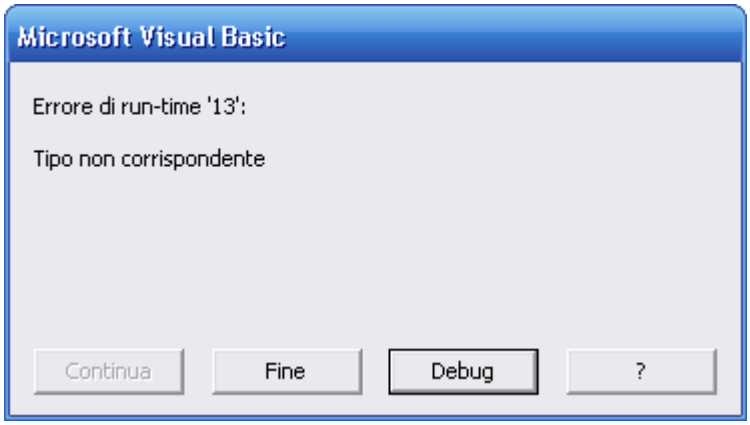

Fig 1

Come si può intuire dal testo si tratta di un errore generato dal codice, premendo sul pulsante **Debug** verrà automaticamente aperto l'editor di VBA e verrà riportata la routine (o Sub) che lo ha causato

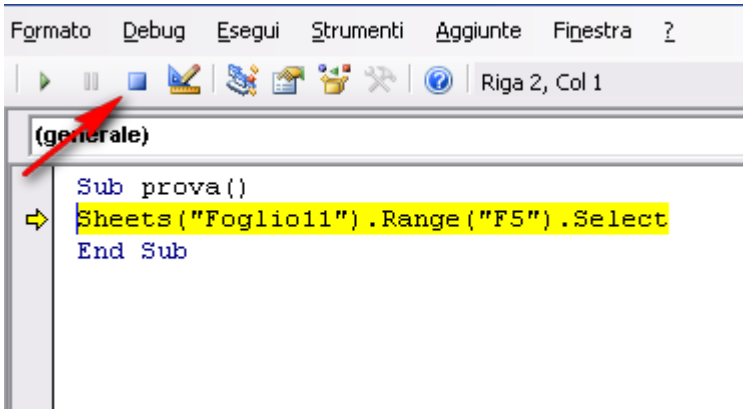

Fig. 2

Nella figura sopra riportata si può notare che la riga di codice che ha causato l'errore è già evidenziata in giallo e questo facilita notevolmente le cose, l'errore è già individuato e si può intervenire e correggere il codice, tuttavia l'errore ha bloccato l'esecuzione della macro e per poter continuare ad operare si deve ripristinare l'editor (cioè sbloccarlo) e in seguito rimediare all'errore. Per eseguire questa operazione basta premere sul pulsante blu in alto nella barra degli strumenti contrassegnato dalla freccia rossa, il quale interrompe il Debug del codice e ripristina l'uso dell'editor.

Spieghiamo ora la natura dell'errore: nella nostra cartella di lavoro, l'errore è stato causato per il semplice fatto che **non esiste** all'interno della stessa un foglio denominato *Foglio11*, per cui VBA ha evidenziato questa situazione con un errore di run-time interrompendo l'esecuzione della macro, in pratica non ha trovato il percorso che gli è stato indicato. Esaminiamo ora il seguente codice:

```
Sub prova()
MsgBox "Ciao Mondo", "Messaggio di saluto"
End Sub
```

Mandando in esecuzione la routine viene rimandato lo stesso avviso del codice precedente

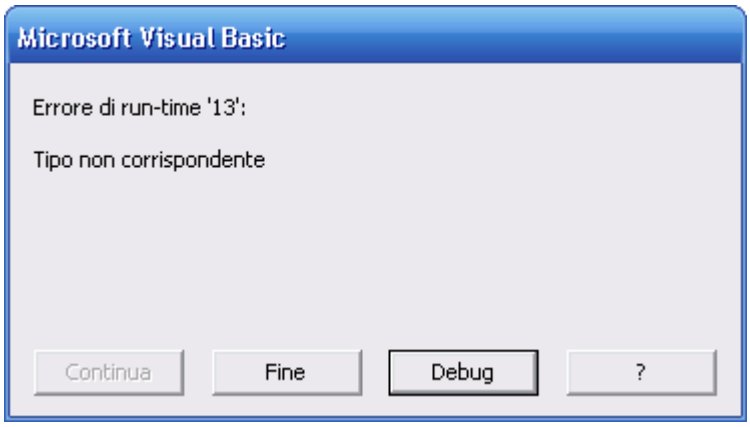

Fig. 3

In questo caso non ci sono errori di sintassi nell'enunciato *MsgBox*, i testi sono correttamente posti tra virgolette, ed è presente la virgola di separazione tra gli argomenti (Ndr. Tutti gli argomenti di MsgBox, salvo il primo, sono opzionali pertanto VBA accetta la presenza di due soli argomenti per MsgBox come sintassi corretta), ma quando VBA cerca di eseguire l'enunciato visualizza una finestra di dialogo di errore come sopra esposta.

Questa finestra di dialogo informa che l'errore si è verificato in fase di esecuzione della procedura e contiene un messaggio che descrive l'errore stesso, in questo caso si tratta di un argomento di tipo errato (non corrispondente). In effetti esaminando la riga, si vede che manca lo spazio che funge da segnaposto per il secondo argomento, *Buttons*, facoltativo. Quando VBA sottopone al Parsing l'enunciato, compila il testo fra virgolette "Messaggio di saluto" come secondo argomento di MsgBox invece di terzo argomento, a causa dell'omissione del segnaposto fra le due virgole. Dato che l'argomento buttons deve essere un numero e non un testo, il VBA segnala che il tipo di dati passato alla procedura MsgBox non è del tipo giusto per quell'argomento.

E' possibile intercettare un errore nel progetto VBA con l'istruzione **On Error** per deviare il flusso del programma verso una routine che gestisca l'errore trasmettendo le istruzioni necessarie per risolverlo. Per poter controllare un errore si inserisce all'inizio della routine l'istruzione *On Error GoTo [etichetta]* e quando si verifica un errore, questa parola chiave interrompe l'esecuzione del codice del programma per "saltare" alla posizione contrassegnata da [etichetta] che deve trovarsi nella stessa procedura che contiene l'istruzione *On Error*.

Per esempio se la routine Command1_Click ha come prima istruzione un *On error goto err* (dove err è il nome dell'etichetta) in caso di errore il flusso del programma va direttamente all'istruzione presente nell'etichetta err, dove si deve inserire il codice o la procedura che tratteranno opportunamente l'errore. Se invece tutto il flusso del programma procede correttamente, l'istruzione On Error viene ignorata e si arriva alla fine della routine. Ci sono vari modi per gestire un errore utilizzando le seguenti istruzioni:

- *On Error GoTo 0 :* Indica al programma di ignorare l'istruzione in caso di errore e il programma continuerà
- *On Error Resume :* Indica che in caso d'errore si ritenta di eseguire l'istruzione che lo ha generato
- *On Error Resume Next :* Indica al programma che in caso d'errore verrà eseguita l'istruzione successiva a quella che ha generato l'errore

E' abbastanza chiaro che solo utilizzando l'istruzione *On Error GoTo [etichetta]* si può trattare l'errore perché negli altri 3 enunciati la situazione si risolve a prescindere dal tipo di errore, in pratica si salta l'errore ma non lo si risolve.

Ad ogni modo è consigliabile porre gli enunciati di gestione degli errori alla fine di ogni procedura inserendo delle funzioni tipo *Exit Sub* oppure *Exit Function*. Vediamo un esempio di codice in un routine con gestione degli errori.

```
Sub Command1_Click()
On Error GoTo err
  MsgBox "Ciao Mondo", "Messaggio di saluto"
Exit_Command1:
Exit Sub

err:
MsgBox Err.Description
Resume Exit_Command1
End Sub
```

Nel codice di esempio sopra esposto *On Error* attiva la gestione degli errori, se si verifica un errore la procedura salta all'etichetta di riga *err* che indica l'inizio della gestione degli errori. La prima riga riporta una finestra di dialogo che visualizza il codice di errore, poi passa all'istruzione *Resume* che ci porta alla routine Exit_Command1 che con l'istruzione Exit Sub ci fa uscire dalla procedura. Se eseguiamo il codice sopra riportato ci riporta un avviso del genere.

Fig. 4

La forma *On Error Goto 0*, è la modalità predefinita in VBA e indica che quando si verifica un errore di runtime VBA dovrebbe visualizzare la sua finestra di messaggio di errore in fase di esecuzione standard, consentendo di immettere il codice in modalità di Debug o di terminare il programma VBA. In pratica avere On Error Goto 0 è come non avere un gestore degli errori attivato, in quanto qualsiasi errore causerà VBA verrà visualizzata la casella di messaggio di errore standard.

La forma*, On Error Resume Next*, è la forma più comunemente usata e abusata, con questa espressione si insegna a VBA di ignorare essenzialmente l'errore e riprendere l'esecuzione sulla riga successiva di codice. E 'molto importante ricordare che On Error Resume Next non risolve in nessun modo l'errore, ma indica semplicemente a VBA di continuare come se non ci fosse stato nessun errore. Tuttavia, l'errore può avere effetti collaterali, come variabili o oggetti impostati su Nothing non inizializzate ed è una responsabilità del codice di verificare una condizione di errore e prendere i provvedimenti opportuni. A tale scopo, vedendo il codice sotto riportato si deve testare il valore di *Err.Number* e se non è zero si deve eseguire la correzione appropriata. Per esempio:

```
On Error Resume Next
   N = 1/0
   If Err.Number <> 0 Then
      N = 1
   End If
```

Questo codice tenta di assegnare il valore 1/0 alla variabile N e questo è un'operazione non ammessa, e VBA genererà un errore 11 (divisione per zero) e dato che abbiamo l'espressione On Error Resume Next, il codice continua, ma in seguito viene verificato il valore di Err.Number e viene assegnato un altro valore alla variabile N

La forma *On Error Goto [etichetta]* indica a VBA di trasferire l'esecuzione alla riga che segue l'etichetta di riga specificata e ogni volta che si verifica un errore, l'esecuzione del codice va subito alla linea che segue l'etichetta di riga e non viene eseguita nessuna istruzione presente nel tra l'errore e l'etichetta, incluse le affermazioni di controllo del ciclo.

```
On Error Goto Err11:
N = 1/0
  'altro codice
Exit Sub
Err11:
 'codice di gestione degli errori
Resume Next
End Sub
```

Il gestore degli errori abilitato e attivo

Un gestore di errori è detto attivato quando un'istruzione On Error viene eseguita e indica il codice che viene eseguito quando si verifica un errore e l'esecuzione viene trasferita in un'altra posizione tramite la dichiarazione On Error Goto [etichetta]. Possiamo definire un blocco di gestione degli errori, chiamato anche gestore di errori, una sezione del codice che prende il controllo del programma attraverso una dichiarazione On Error Goto [etichetta]. Questo codice deve essere progettato sia per risolvere il problema e riprendere l'esecuzione nel blocco di codice principale o di interrompere l'esecuzione della procedura, non è possibile utilizzare On Error Goto [label] semplicemente per "saltare" tra le linee di codice. Ad esempio, il seguente codice non funzionerà correttamente:

```
On Error GoTo Err1:
Debug.Print 1/0
 'altro codice
Err1:
On Error GoTo Err2:
Debug.Print 1/0
 'altro codice
Err2:
```

Quando si verifica il primo errore, il flusso del programma viene trasferito alla linea Err1 ma l'errore è ancora attivo quando si verifica il secondo errore, e quindi quest'ultimo non viene intercettato dalla istruzione On Error.

L'istruzione Resume

La dichiarazione **Resume** indica al VBA un punto specifico del codice dove riprendere l'esecuzione, inoltre è possibile utilizzare Resume solo in un blocco di gestione degli errori, qualsiasi altro uso causerà un errore. Si deve considerare che *Resume* è l'unico modo, a parte uscire dalla procedura, per uscire da un blocco di gestione degli errori. Si deve ricordare di **non** utilizzare la dichiarazione *Goto* per dirigere l'esecuzione del codice da un blocco di errore, agire in questo modo si causeranno strani problemi ai gestori di errori. La dichiarazione Resume può assumere tre forme sintattiche

- Resume
- Resume Next
- Resume [Etichetta]

Usato da solo, Resume provoca l'esecuzione di riprendere alla riga di codice che ha causato l'errore, in questo caso è necessario assicurarsi che il blocco di gestione degli errori risolva il problema che ha causato l'errore iniziale, in caso contrario, il codice entrerà in un ciclo infinito, saltando tra la riga di codice che ha causato l'errore e il blocco di gestione degli errori. Il codice seguente tenta di attivare un foglio di lavoro che non esiste, questo provoca un errore, e il codice salta al blocco di gestione degli errori che crea il foglio, correggere il problema, e riprende l'esecuzione alla riga di codice che ha causato l'errore.

```
On Error GoTo Err1:
    Worksheets("Foglio11").Activate
    Exit Sub

    Err1:
    If Err.Number = 9 Then
        'il foglio non esiste, quindi lo crea
        Worksheets.Add.Name = "Foglio11"
      'torna alla riga di codice che ha causato il problema
        Resume
    End If
```

La seconda forma di Resume è **Resume Next**, che riprende l'esecuzione del codice nella riga immediatamente successiva alla riga che ha causato l'errore. Il seguente codice genera un errore (11 - divisione per zero) quando si tenta di impostare il valore di N. Il blocco di gestione degli errori assegna il valore 1 alla variabile N, e poi riprende l'esecuzione del codice con l'istruzione dopo l'istruzione che ha causato l'errore.

```
On Error GoTo Err1:
    N = 1 / 0
    Debug.Print N
    Exit Sub

    Err1:
    N = 1
    'torna alla riga successive che ha causato l'errore
    Resume Next
```

La terza forma di Resume è **Resume [etichetta]** che porta l'esecuzione del codice a riprendere in un'etichetta di riga, questo permette di saltare una sezione di codice se si verifica un errore.
Per esempio:

```
On Error GoTo Err1:
    N = 1 / 0
      'codice che viene saltato se si verifica un errore

    Label1:
    'codice da eseguire
    Exit Sub

    Err1:
    'ritorna alla linea Label1
    Resume Label1:
```

Ogni procedura non deve necessariamente avere un codice di gestione degli errori quando si verifica un errore, VBA utilizza l'ultima istruzione On Error per dirigere l'esecuzione del codice, tuttavia, se la procedura in cui si verifica l'errore non ha un gestore degli errori, VBA guarda indietro attraverso le chiamate di procedura che portano al codice errato. Ad esempio, se la procedura A chiama B e B chiama C, e A è l'unica procedura con un gestore degli errori, se si verifica un errore nella procedura C, l'esecuzione di codice viene immediatamente trasferito al gestore di errore nella procedura A, saltando il codice rimanente in B