Variant variabili

http://www.vbsimple.fbi/weird/wrd_07.htm

Difficoltà: 2 / 5

Questa stranezza mi è capitata mentre sviluppavo un programma in Delphi e solitamente programmando in Visual Basic non ci si rende conto del difetto. L'articolo utilizza la libreria ADO per connettere un Recordset, quindi salvarlo su file e ricaricarlo dallo stesso.

Utilizzeremo soltanto un form e tutto il codice sta nella routine di gestione dell'evento ***Load***:

```
 1. Option Explicit
 2.
 3. Private Sub Form_Load()
 4.     Dim conn As ADODB.Connection
 5.     Dim rs As ADODB.Recordset
 6.
 7.     Set conn = New ADODB.Connection
 8.     Call conn.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & App.Path &
    "\db1.mdb")
 9.     Set rs = conn.OpenSchema(adSchemaColumns, Array(Empty, Empty, "PROVA", Empty))
10.     With rs
11.         MsgBox "Test 1 => " & VarType(.Fields("NUMERIC_PRECISION").Value)
12.         MsgBox "Test 2 => " & .Fields("NUMERIC_PRECISION").Type
```

Le righe 4 e 5 dichiarano rispettivamente una Connessione ed un Recordset, mentre le righe 7 e 8 istanziano ed aprono la connessione al database Access DB1.MDB posto nella stessa cartella del progetto.

La riga 9 legge lo schema Columns che restituisce un Recordset contenenti tutte le colonne della tabella selezionata (di nome **PROVA**). Per una spiegazione del funzionamento di *OpenSchema* si rimanda alla guida di ADO, perchè inutilmente lunga da trattare in questo articolo. La stessa documentazione dice che il recordset restituito conterrà una colonna di nome **NUMERIC_PRECISION** e di tipo DBTYPE_UI2, corrispondente ad un *Unsigned Integer da 2 bytes* e riprodotto in ADO come adUnsignedSmallInt, dal valore 18.

Alla riga 11 è effettuato questo controllo, inizialmente sul valore dato dalla proprietà ***Value*** e quindi come controprova sulla proprietà ***Type***. Secondo quanto dice la guida entrambi i test dovrebbero visualizzare il valore 18. Alla prova pratica purtroppo non é così:

Figura 1

Figura 2

Il primo dei test, visualizzato nella **Figura 1** mostra come il tipo Variant del valore

restituito sia 3, corrispondente a vbLong ed alla sua trasposizione in ADO con adInteger, *Intero con segno da 4 bytes* ovvero DBTYPE_I4. Il secondo test, mostrato in **Figura 2**, dimostra come il tipo di dati che ADO dice di restituire invece è il regolare 18 adUnsignedSmallInt ovvero DBTYPE_UI2.

Di primo acchitto si potrebbe accettare la stranezza con l'alibi che Visual Basic non è in grado di gestire un tipo di dati Intero senza segno e pertanto forza automaticamente la conversione al tipo successivamente più ampio, come Long. Il successivo test dimostrerà come non solo la ragione non può esser questa ma ne avverrà un'altra ancora.

```
13.          .Save "c:\wrd_07.xml", adPersistXML
14.          .Close
15.          Call .Open("c:\wrd_07.xml", Nothing)
16.          MsgBox "Test 3 => " & VarType(.Fields("NUMERIC_PRECISION").Value)
17.          MsgBox "Test 4 => " & .Fields("NUMERIC_PRECISION").Type
```

Dalla riga 13 il Recordset aperto verrà salvato su un file XML di nome WRD_07.XML, chiuso e riaperto ricaricando lo stesso file salvato. Successivamente sono effettuati i due test già eseguiti in precedenza (righe 16 e 17). Questa volta però la dimostrazione cambia:

Figura 3

Figura 4

Entrambi i test restituiscono gli stessi dati: il tipo Variant dato dalla proprietà Value è 18, corrispondente a DBTYPE_UI2, lo stesso tipo che la proprietà Type dice di restituire. Questo dimostra anche l'inaccettabilità della tesi sulla predetta conversione automatica; Visual Basic infatti forzerà la conversione a Long soltanto nel momento in sarà obbligato a farlo, ad esempio quando il numero da rappresentare non rientrerà nel tipo Integer nativo di VB. Il linguaggio è perfettamente in grado di maneggiare numeri dichiarati con il tipo 18: *Intero senza segno e da 2 bytes*.

A questa stranezza se ne aggiunge un'altra, data dalla variazione del tipo di Variant restituito dalla proprietà Value; il primo test restituiva valore 3 (DBTYPE_I4) mentre il secondo valore 18 (DBTYPE_UI2). In quesat seconda analisi dopo il ricaricamento del Recordset entrambi i test restituiscono valore 18 (DBTYPE_UI2), quello corretto ed in linea con la documentazione di ADO.

```
18.          .Close
19.      End With
20.      conn.Close
21.      Kill "c:\wrd_07.xml"
22.      Set rs = Nothing
23.      Set conn = Nothing
24.      Unload Me
25.  End Sub
```

Segue il codice occupato della chiusura del Recordset, della connessione, del form e della cancellazione del file, che non richiede alcuna spiegazione.

Nessuna colpa può essere addossatta a Visual Basic riguardo queste due stranezze ma il comportamento è totalmente a carico di ADO; ne è prova il fatto che questa dimostrazione

è stata effettuata come controprova del medesimo malfunzionamento verificatosi inizialmente in un progetto Delphi.

Fibia FBI
20 Dicembre 2003


Torna all'indice delle Stranezze

