La programmazione del Mac
Le Risorse (2)

di Raffaello De Masi

L'altra volta abbiamo parlato in generale di risorse, primo mattone della programmazione Macintosh. Abbiamo anche detto, che nell'ambito del sistema operativo, esiste un sottosistema, operativo anch'esso, che è destinato al maneggio delle risorse. Fortunatamente, tutti i linguaggi, dal più scalcinato Basic al Pascal, al più raffinato dei C, possiedono librerie di routine all'uopo costruite adatte a maneggiare adeguatamente i più completi pacchetti di risorse. Inoltre, proprio perché il maneggio delle risorse, nella quasi totalità delle sue funzioni, si limita a una chiamata e ad un passaggio di parametri, praticamente l'uso di queste routine è praticamente identico in tutti i linguaggi. E così, guarda caso, utilizzeremo ancora una volta il Basic (QBasic, ZBasic o TrueBasic) per gli esempi che forniremo

L'utilizzo delle risorse

Probabimente il QuickBasic di Microsoft (e, ovviamente, il MS-Basic che lo ha preceduto) ha la tecnica più facile di gestione delle risorse. In esso le risorse sono tutte contenute nel resource fork del file di codice per cui è sufficiente che l'interprete apra quelle che gli servono al momento opportuno. True Basic e ZB invece, (e utenti QB più sosfisticati, che non vogliono sistemare le loro risorse nel codice iniziale), devono invece obbligatoriamente aprire un resource file. Come avviene ciò? Con delle semplici chiamate del tipo:

```
[QuickBasic]
OpenResFile Filename$, ResRef

[ZBasic]
ResRef = FN OPENRESFILE(FileName$)
```

Come si vede le implementazioni sono piuttosto simili e forniscono, come risultato, il numero di riferimento di file del resource file aperto. È la prima pietra della utilizzazione delle risorse. Ne riparleremo tra poco.

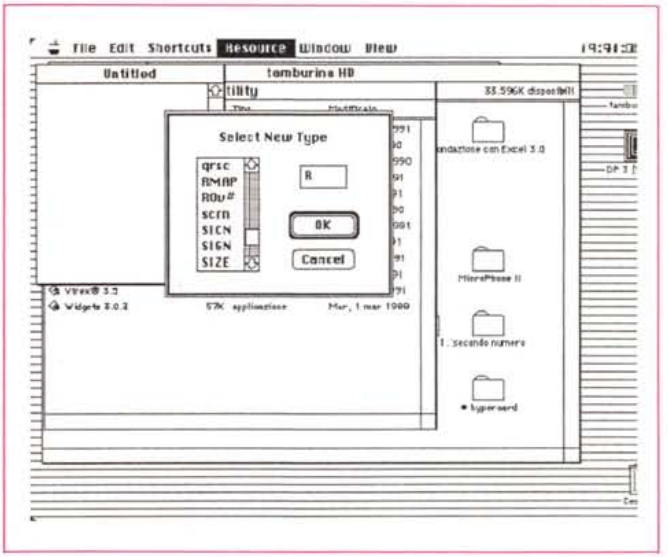
Una routine per così dire universale per il richiamo e il maneggio di un file risorse potrebbe essere:

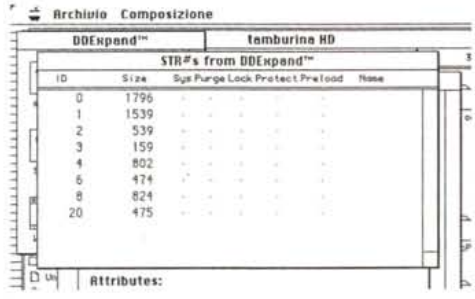
```
Hndl& = FN GETRESOURCE (CVI("Mc"),0)
LONG IF Hndl& = 0
ResRef = FN OPENRESFILE ("MyResFile")
END IF
```

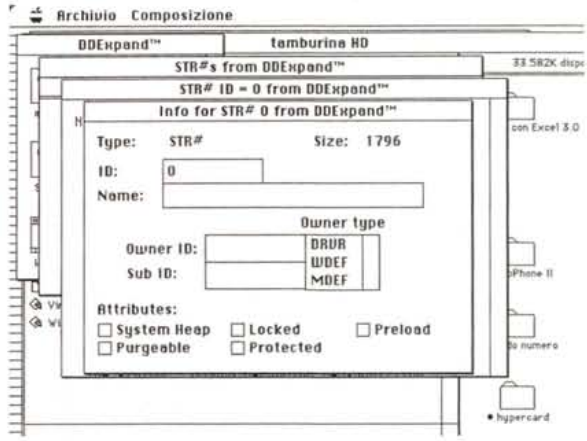
Si tratta della chiamata a un file di risorse che, avendo valore numerico zero, esiste solo come set di risorse proprie del programmatore; in altri termini la routine «chiama» un file RES di nome Mc o, ovviamente, qualunque altro nome assegnato dal programmatore, con un ID number pari a [0].

Se il programma riesce a trovare questo file, ricerca coronata da successo se una handle non pari a zero è restituita, si tratta ovviamente di una applicazione già compilata, e, così, non c'è necessità di «aprire» un file risorse esterno. Se l'handle è invece pari a zero, la risorsa (o il set di risorse) con chiamata a un file esterno ha bisogno di essere aperto (ovviamente, come abbiamo già detto in precedenza, questo avviene solo in ZBasic (e più generalmente in linguaggi che adottano file esterni).

La cosa, così com'è, non è eccessivamente complessa da manipolare; ma c'è una scorciatoia efficiente e interessante, che merita di essere menzionata. Esiste un comando, specifico di QB, SYSTEM(4), che punta direttamente al ResFile sia in maniera compilata che interpretata. Un esempio di routine d'u-







so del comando potrebbe essere:

```
AA = SYSTEM(4)
if AA = 0 then OpenResFile RFile$, Ref
Get ResRef, Type$, ResID, Hndl&
```

ma, in ogni caso, il codice precedentemente esposto diviene superfluo se si è sistemato un resource fork direttamente nel codice. Bisogna solo ricordare, e ciò è importantissimo, che occorre chiudere adeguatamente il ResFile prima dell'END del programma, attraverso la semplice istruzione:

- CloseResFile ResRef
- CALL CLOSERESFILE(ResRef)

la prima delle quali si riferisce al QB e la seconda allo ZB.

La chiamata alle risorse

Apple ha creato per il Macintosh quel miracolo che è il toolbox. E nell'ambito del toolbox il Resource Manager è davvero il genio nella scatola che permette il maneggio disinvolto delle risorse stesse. Per usare un gergo fiabesco, basta solo invocarlo (che coincidenza di termini!) ed esso risponde.

Per maneggiare una risorsa, occorre avere a disposizione una handle, un aggancio. Una handle (letteralmente maniglia, termine che probabilmente meglio di tutti rende l'idea) è, come probabilmente molti sanno, un indirizzo ad un entry point in una lista di indirizzi stessi (in gergo, più semplicemente, è un puntatore a un puntatore principale — di una lista —). Il Macintosh, come molte altre macchine, utilizza questo tipo di puntamento indiretto in quanto i dati in memoria «fluttuano» a seconda della utilizzazione della memoria stessa (per conoscere l'indirizzo attuale di un dato, è sufficiente adottare la chiamata PEEK LONG(Hndl&), chiamata comunque inutile visto che l'indirizzo può cambiare in ogni momento). Nel caso, addirittura, che l'handle non è in memoria, il Memory Manager esegue un caricamento da disco e restituisce l'handle stessa.

La funzione FN GETRESOURCE, presente in ZBasic è una diretta implementazione della chiamata al toolbox _GetResource (se si dà una semplice occhiata alla chiamata _GetResource in Inside Macintosh si vedrà immediatamente che i parametri inviati e ricevuti sono gli stessi di quelli maneggiati da ZB). La cosa non è da poco se si considera che è possibile usare la documentazione Apple per ovviare alla scarsità di informazioni reperibili nel manuale dello ZBasic.

QB invece è molto più efficiente e, per certi versi sicuro. Infatti poiché generalmente sono aperti contemporaneamente diversi Resource File insieme (ad esempio quello del System, quello dell'applicazione corrente, e magari quello della stampante e di qualche DA, senza sconfinare nel Multifinder e nella selva operativa del System 7), QB impone al programmatore di indicare su quale file desidera lavorare; in altri termini, paradossalmente, il programmatore è difeso contro se stesso.

QB inoltre impone di passare una stringa, come parametro, per specificare il tipo di risorsa adottato. In altri termini, il toolbox reclama un long integer; e in questo c'è ancora d'aiuto Apple, che ha creato le risorse [rez] convertendo direttamente la rappresentazione ASCII (ad esempio TEXT) in un long integer. In ZB, invece, occorre eseguire manualmente la conversione, ancora una volta attraverso un comando, del tipo:

```
Hndl& = FN GETREOURCE(CVI("Mc"),0)
```

con un comando in più, che però ha molto più senso a livello di programmazione.

Ma non sempre è così: ad esempio se si desidera maneggiare una resource del tipo PICT non è più così complicato. È sufficiente eseguire una chiamata al toolbox _DrawPicture, e _GetResource esegue tutto quello di cui si ha bisogno.

Molto spesso, però, nel caso di una programmazione più raffinata, può capitare di dover spostare dati di risorse in strutture proprie del linguaggio che stiamo utilizzando. Questo avviene seguendo esattamente una serie successiva di ordini, così strutturati:

- chiudere il resource handle;
- ricavare l'indirizzo attuale della risorsa attraverso il già visto comando PEEKLONG(Hndl&);
- eseguire un indirizzamento dei dati BASIC con una funzione del tipo VARPTR(F9) spostare il blocco di dati (BLOCKMOVE) dalla risorsa alla struttura BASIC;
- aprire la risorsa;
- eseguire una chiamata alla funzione _ReleaseResource, per forzare il resource Manager a estrarre i dati di risorsa dalla memoria. Ovviamente questa operazione va effettuata solo alla fine delle operazioni.

Un po' di attenzione

Le risorse sono facili, e anche piacevoli da maneggiare, e permettono di potenziare in maniera avanzata l'applicazione che si sta scrivendo; ma, come in

tutte le cose troppo facili da realizzare, occorre fare attenzione ad alcuni piccoli particolari che, se trascurati, possono rendere davvero difficile la vita.
In particolare occorre:

- evitare di eseguire spostamenti di blocchi di memoria utilizzando gli stessi che si sta usando. È molto più semplice e sicuro affidare tutto al Resource manager (sta lì per questo!) adottando la chiamata a _ReleaseResource.
- Non creare risorse con un nome formato tutto di lettere minuscole (esempio «roma»); Apple riserva questa convenzione a sé, e al proprio uso; potrebbe succedere che una semplice nuova release di sistema porti a una sovrapposizione di risorse, con bombe e blocchi di cui ben difficilmente poi riusciremmo a capire il motivo. Allo stesso modo evitare di usare un ID number inferiore a 128; anche questi sono riservati ad Apple.

Per concludere

Una delle risorse più utilizzate è la string list, di cui abbiamo già avuto modo di parlare (il tipo è STR#). Giusto per fare un esempio proviamo a crearne una.

Lanciamo ResEdit e selezionaiamo New dal file menu. Se si possiede la versione 2.1 (vedi figure), occorre assegnare già adesso un nome al file. La fase successiva è quella di creare una nuova risorsa usando il comando COMMAND-K. Si presenterà una lista da cui sarà possibile scegliere. Scegliamo, appunto STR#. Si presenterà una finestra (vedi figure) che permetterà l'input del nome della stringa stessa. Scegliamo un nome e completiamo l'operazione. La fase successiva è quella di assegnare un ID Number, ad esempio 250 e scegliamo gli attributi necessari, tra cui il Preload e il Purgeable, che fanno parte dei «resource attributes».

La prerogativa del «cancellabile» è quasi sempre richiesta, infatti, è presente in quasi tutte le applicazioni. Questo permette al Resource manager di disporre di memoria virtuale, se ce ne fosse necessità, cancellando la risorsa dalla memoria in caso di necessità. Un esempio di risorsa, invece, non cancellabile è il programma stesso e certi blocchi di dati assolutamente irrinunciabili.

L'altro attributo, nominato, è il «Preload», che permette di caricare immediatamente in memoria la risorsa se ce ne fosse necessità. La contropartita di tutto ciò è rappresentata da un più lungo tempo di startup, e un più esteso uso della memoria (in effetti le Preload sono risorse, presenti nel programma, che vengono solo caricate se c'è disponibilità di memoria; esse vengono scartate quando, ad esempio apriamo un programma e incontriamo il messaggio «Occorrono XXXX byte di memoria e cene sono solo YYYY; vuoi aprire il programa lo stesso?»: in questo caso molti «Preload» non vengono caricati, in base all'ID Number). L'unica cosa da tenere a mente è ovviamente il nome della risorsa creata. Il resto è automatico.

E così abbiamo terminato il discorso di massima che ci eravamo preposto riguardo alle risorse. Ma prossimamente avremo modo di continuarlo.

MC