Costruire un file di help

2ª parte

Dopo un'introduzione generale sulla costruzione di un file di guida, addentriamoci ora nell' analisi approfondita delle macro

di Roberto Scano

Come avevo già accennato nella prima parte dell'articolo, nella sezione [CONFIG] possono essere inserite delle macro proprie di Winhelp, di cui darò una breve descrizione ed esemplificazione nei paragrafi successivi. Queste macro sono necessarie al "creatore" del file di help per poter personalizzare al massimo il file HLP, aggiungendo o rimuovendo menu, pulsanti o richiami a finestre secondarie, eseguibili Windows o funzioni da DLL.

Inizierò con una suddivisione degli argomenti a seconda dell'utilizzo nel seguente modo:

- Gestione dei bottoni
- Gestione dei menu e dei bookmark
- Gestione delle finestre
- Macro di JUMP e POP-UP
- Utilizzo di DLL

GESTIONE DEI BOTTONI

Le macro descritte in questo paragrafo servono per la gestione dei push button della button-bar di Winhelp. I bottoni sono consigliati per aggiungere delle macro necessarie all'utente in più argomenti, come il bottone di stampa e di impostazione segnalibro. Per poter aggiungere un bottone alla button-bar è necessario crearlo, tramite la macro

```
CREATEBUTTON ("button-id", "name",
    "macro")
```

dove *button-id* rappresenta un nome che Winhelp utilizza per identificare il bottone, *name* è il testo che appare sul bottone (come per qualsiasi controllo Windows, per assegnare un tasto di attivazione è necessario utilizzare l'ampersand "&" prima del carattere desiderato) e *macro* è nome della macro definita dall'utente o macro di Winhelp che viene eseguita alla pressione del bottone. Ad esempio, per creare un pulsante che esegua il programma MIO.EXE e che abbia come testo "Esegui", sarà necessario scrivere:

```
CREATEBUTTON ("btn_app", "&Esegui",
    "execprogram ('MIO.EXE'), 0")
```

La macro EXECPROGRAM ("linea_di_comando", visualizzazione) esegue l'applicazione indicata in *linea_di_comando* nella modalità stabilita da *visualizzazione*. Il valore di *visualizzazione* è un intero che può assumere i seguenti valori: 0 (normale), 1 (l'applicazione è ridotta ad icona) oppure 2 (l'applicazione è massimizzata). Un esempio di questa macro è presente in forma sorgente nella figura 1).

Come per quasiasi bottone, anche il nostro può essere abilitato, disabilitato o eliminato definitivamente. Per ognuna di queste tre operazioni è necessaria una stringa, corrispondente al *button*-id prestabilito nella fase di creazione del bottone appena vista. I comandi che effettuano queste operazioni sono i seguenti:

DISABLEBUTTON (" *button-id*")
disabilita un bottone
ENABLEBUTTON (" *button-id* ")
riabilita un bottone disabilitato in precedenza dalla macro DISABLEBUTTON
DESTROYBUTTON (" *button-id*")
rimuove un bottone dalla button bar di Winhelp

Quindi, usando come esempio il pulsante creato in precedenza, possiamo disabilitarlo con la macro DISABLE BUTTON ("btn_app") ; riabilitarlo con la macro ENABLEBUTTON ("btn_app") mentre per eliminarlo è necessaria la macro DESTROYBUTTON ("btn_app") . È inoltre possibile modificare la macro associata ad un bottone, sia esso creato dall'utente che un button di default di Winhelp. Il comando da utilizzare per far ciò è il seguente:

```
CHANGEBUTTONBINDING ("button_id",
    "button_macro")
```

dove *button_macro* è nome della macro definita dall'utente o macro di Winhelp che viene eseguita alla pressione del bottone, e *button_id* rappresenta un nome che Winhelp utilizza per identificare il bottone. Se si tratta di un bottone creato dall'utente, *button-id* assume il valore indicato nella macro CREATEBUTTON mentre se si tratta di uno bottone standard di Winhelp, i valori possibili sono i seguenti:

BTN_CONTENTS
per il bottone "Sommario"
BTN_SEARCH
per il bottone "Cerca"
BTN_BACK
per il bottone "Precedente"
BTN_PREVIUS
per il bottone "<<"
BTN_NEXT
per il bottone ">>"
BTN_HISTORY
per il bottone "Cronologia"

Quindi, se si desidera assegnare al pulsante che abbiamo creato una nuova macro di esecuzione (per il programma MIO2.EXE) bisogna utilizzare la seguente macro:

```
CHANGEBUTTONBINDING ("btn_app", ",
    "execprogram ('MIO2.EXE'), 0")
```

Un'altra macro riguardante i pulsanti è BROWSEBUTTONS( ) , già vista nell'articolo precedente.

GESTIONE DEI MENU E DEI BOOKMARK

Tramite le macro che analizzeremo in questo paragrafo è possibile avere la piena gestione dei menu, essendoci la possibilità di creare, inserire, aggiungere, eliminare, abilitare, disabilitare, modificare i menu. Iniziamo con la macro per la creazione di un nuovo menu:

```
INSERTMENU ("menu_id", "menu_name",
    menu_position)
```

dove menu_id rappresenta un nome che Winhelp utilizza per identificare il button, *menu_name* è il testo che verrà visualizzato nella barra dei menu e *menu_position* è la posizione nella quale verrà visualizzato il menu. I valori assunti partono da 0 e sono crescenti da sinistra verso destra. Ad esempio, se desideriamo creare un nuovo menu chiamato Menu 1 e vogliamo posizionarlo all'estrema sinistra della finestra di guida, è necessario eseguire la seguente macro:

```
INSERTMENU ("esempio", "&Menu 1", 0)
```

Oltre ad un menu è possibile inserire anche le singole voci di sottomenu: vi sono due possibilità di inserimento:

```
APPENDITEM ("menu_id", "item_id",
    "item_name", "macro")
INSERTITEM ("menu_id", "item_id",
    "item_name", "macro", position)
```

la differenza tra le due macro sta nel fatto che la prima permette soltanto di aggiungere un elemento ad un menu esistente (sistemandolo in ultima posizione) mentre con la seconda macro vi è la possibilità di decidere dove posizionare il nuovo menu. A parte il parametro *position*, il significato degli argomenti delle due macro è identico. *Menu_id* rappresenta un nome che Winhelp utilizza per identificare il menu; può essere un valore definito dalla macro INSERTMENU oppure un valore standard di Winhelp; in questo secondo caso, i valori possibili sono i seguenti:

MNU_FILE
per il menu "File"
MNU_EDIT
per il menu "Modifica"
MNU_BOOKMARK
per il menu "Segnalibro"
MNU_HELPON
per il menu "Guida"

il parametro *item_id* è il valore utilizzato da Winhelp per identificare la nuova voce di menu; *item_name* è il testo del menu; *macro* rappresenta la macro di Winhelp oppure definita dall'utente che dovrà essere eseguita quando viene selezionata la voce di menu in questione. Infine *position*, disponibile sono in INSERT MENU, è la posizione nella quale verrà visualizzato il menu; i valori assunti partono da 0 per la prima posizione di menu. Se ad esempio vogliamo creare una sottovoce di menu al nostro "Menu 1" chiamata "Sottomenu 1" e vogliamo fargli attivare l'About Box, dovremmo scrivere la seguente macro:

```
INSERTITEM("esempio", "esempio2",
    "&Sottomenu1", "about( )", 1)
```

Come accade per i bottoni, è possibile abilitare, disabilitare o eliminare una voce di menu. Per ognuna di queste tre operazioni è necessaria una stringa, corrispondente al *menu_id* stabilito al momento della creazione del menu. Le macro messe a disposizione da Winhelp sono:

```
ENABLEITEM ("item_id")
DISABLEITEM ("item_id")
DELETEITEM ("item_id")
```

È inoltre possibile modificare la macro associata ad un menu, sia esso creato dall'utente che un menu di default di Winhelp, usando il seguente comando:

```
CHANGEITEMBINDING ("item_id", "macro")
```

dove *macro* è il nome della macro definita dall'utente o macro di Winhelp che viene eseguita una volta scelta la voce di menu e *item_id* rappresenta un nome che Winhelp utilizza per identificare una voce di menu. Se si tratta di una voce di menu creata dall'utente, *item_id* assume il valore indicato nella macro APPENDITEM mentre se si tratta di un menu di Winhelp, i valori possibili sono i seguenti:

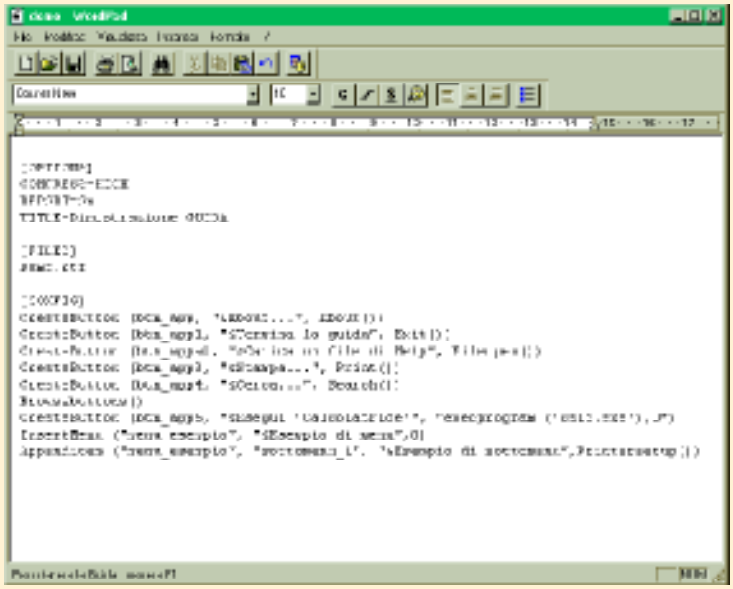
MNU_FILE
per il menu "File"
MNU_EDIT
per il menu "Modifica"
MNU_BOOKMARK
per il menu "Segnalibro"
MNU_HELPON
per il menu "Guida"

Ad esempio, se si desidera assegnare alla voce di sottomenu che abbiamo creato una nuova macro di esecuzione (per il programma MIO2.EXE) bisogna utilizzare la seguente macro:

```
CHANGEITEMBINDING ("esempio2", ",
    "execprogram ('MIO2.EXE'), 0")
```

Le funzioni dei pulsanti e dei menu di sistema possono essere richiamate tramite macro. Questa opzione può essere uti-

Figura 1



Il sorgente di un file HPJ

le, ad esempio, per la stampa di un argomento: sarà sufficiente richiamare la macro di stampa ed assegnarla ad un pulsante il quale, essendo sempre presente nella finestra, offre un risparmio di tempo all'utente. Poiché queste macro possono essere utilizzate sia dai menu che dai button, descriveremo le funzioni in generale. La tabella 1 riporta le macro in ordine alfabetico; notate che esse non hanno bisogno di parametri aggiuntivi, visto che effettuano un semplice richiamo ad una funzione della Guida. A titolo di esempio, se si desidera creare un pulsante per uscire dalla Guida avente come etichetta "Esci", si potrà utilizzare la seguente macro:

```
CreateButton ("btn_uscita", "&Esci",
    "Exit ()")
```

Iniziamo ora ad analizzare la gestione dei segnalibro (bookmark). Come per gli oggetti definiti in precedenza, anche i bookmark hanno una loro fase di creazione, aggiunta, eliminazione, oltre ad altre macro specifiche come il salto ad un determinato segnalibro o l'esecuzione di una istruzione se esiste o meno un determinato segnalibro. La prima operazione da fare è visualizzare la finestra di definizione segnalibro, tramite la macro BookMarkDefine() . Quando l'utente definisce più di 9 segnalibro, nel menu Segnalibro, oltre ai primi 9 segnalibro, viene visualizzata la voce "Altro...", la quale è richiamabile con la macro BookMarkMore() . Se invece vogliamo che venga salvato un nuovo segnalibro, è necessario usare la macro SaveMark ("marker-text") , dove *marker-test* è una stringa di caratteri che si vuole salvare come Segnalibro, per permettere all'utente in futuro di richiamare la presente pagina di guida semplicemente selezionandola tra i segnalibro. Ad esempio, per creare un pulsante che aggiunga un segnalibro "Prova segnalibro" è si deve eseguire la seguente macro:

```
CreateButton ("bkmk", "&Aggiungi
    Segnalibro", "SaveMark (`Prova
    Segnalibro')")
```

Potrebbe anche presentarsi la necessità di rimuovere un segnalibro dalla lista; in questo caso è necessario l'utilizzo della macro DeleteMark("marker_text") ; se il segnalibro non esiste, Winhelp visualizzerà il messaggio d'errore "Argomento non trovato". È inoltre possibile "saltare" ad un determinato argomento tramite uno specifico segnalibro, utilizzando la macro GoToMark("marker_text") . Poiché non sempre si può essere sicuri dell'esistenza di un segnalibro, vi sono delle macro aggiuntive predisposte al controllo dell'esistenza o meno di un segnalibro; se il segnalibro esiste esse eseguono una funzione, altrimenti eseguono qualche altra azione; ad esempio, se il richiamo al segnalibro era effettuato da un determinato button, si può disabilitare il button tramite la macro DISABLEBUTTON. La prima macro da analizzare è IsMark ("marker_text") , la quale restituisce TRUE se "marker_text" esiste nell'elenco dei segnalibro, FALSE in caso contrario. Le due macro seguenti utilizzano la macro IsMark per eseguire a loro volta delle macro:

```
IfThen(IsMark("marker_text"), "macro1")
IfThenElse(IsMark("marker_text"),
    "macro1", "macro2")
```

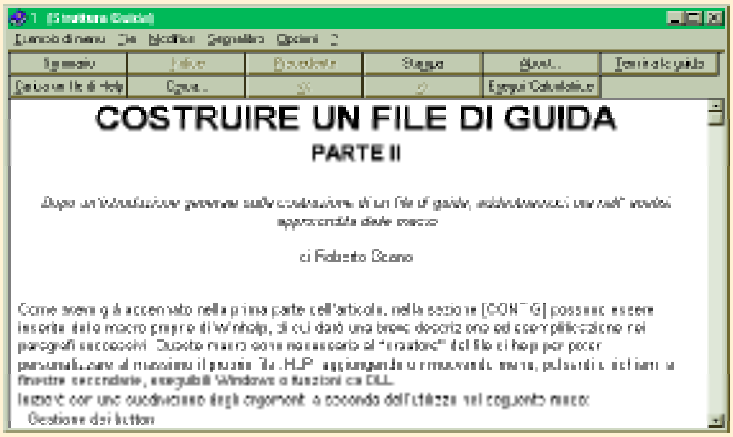
La prima macro esegue "macro1" soltanto se esiste un segnalibro indicato dalla stringa "marker_text"; in caso contrario non esegue nessuna istruzione. La seconda macro esegue "macro1" se esiste il segnalibro indicato dalla stringa "marker-text" e, in caso non esista, esegue "macro2". Se, ad esempio, volessimo abilitare un button ed assegnargli un salto al segnalibro creato in precedenza soltanto se quest'ultimo esiste effettivamente, e disabilitare il pulsante in caso contrario, dovremmo eseguire la seguente macro:

```
IfThenElse(IsMark("Prova Segnalibro"),
    "EnableButton (`btn_app')",
    "DisableButton (`btn_app')")
```

Per invertire il significato di un test è sufficiente anteporre NOT alla macro. Ad esempio, se si vuole disabilitare un pulsante se un segnalibro risulta non esistere, è possibile utilizzare la seguente macro:

```
IfThen(Not(IsMark("Prova Segnalibro")),
    "DisableButton(`btn_app')")
```

Figura 2



Il file HLP derivante dal file di progetto mostrato in figura 1

GESTIONE DELLE FINESTRE

Il nostro file di guida può anche essere composto da più finestre; queste vengono indicate nella sezione [WINDOWS] del file di progetto (HPJ), nel punto in cui viene definito il titolo della finestra, il nome di riconoscimento ("Window_name") e la sua posizione.

Alla finestra principale viene riservata la stringa "main" come Window_name mentre alle altre finestre, dette anche finestre secondarie, può essere assegnato un nome qualsiasi.

Le finestre vengono richiamate tramite la macro FocusWindow ("Window_Name") ; con tale comando viene dato il "focus" alla finestra indicata dalla stringa "Window_Name". Di solito - ad esempio i file di Help di Winword, Excel, ecc. - le finestre secondarie vengono aperte in determinate posizioni dello schermo, in modo da non oscurare la finestra principale della Guida. A questo serve la macro seguente:

```
PositionWindow (X, Y, Width, Height,
    WindowState, "Window_Name")
```

dove *X* e *Y* sono le coordinate dell'angolo in alto a sinistra della finestra, *Width* e *Height* impostano la larghezza e l'altezza della finestra, *WindowState* rappresenta lo stato della finestra e può assumere i valori 0 (normale) oppure 1 (massimizzata, nel qual caso gli argomenti precedenti sono ignorati); *window_name* è il nome della finestra indicato nella sezione [WINDOWS] del file di progetto. Ad esempio, se desideriamo creare una finestra secondaria (a cui abbiamo assegnato il nome di "dettagli") nell'angolo superiore sinistro e di dimensioni 640 x 200, occorrerà eseguire la seguente macro:

```
PositionWindow (0, 0, 640, 200,
    "dettagli")
```

MACRO DI JUMP E POP-UP

L'Help Compiler permette inoltre di "saltare" in determinate posizioni all'interno di altri file di Guida, oppure può visualizzare argomenti prelevati da altri file di guida in una finestra pop-up. Iniziamo con l'analizzare le macro di jump, utilizzabili per richiamare un argomento da un altro file di guida allegato alla nostra applicazione, il che rende possibile, ad esempio, creare due file di help di cui uno con le istruzioni e uno con una esercitazione e permettere all'utente di richiamare la guida dall'esercitazione o viceversa. Se si desidera saltare direttamente al sommario del file di help prescelto è sufficiente eseguire la macro JumpContents ("Filename") ; se invece si desidera saltare ad un punto specifico del file, conoscendo il suo context-number, occorrerà usare la macro JumpContext ("Filename", context_number) . Il valore di *context_number* è visibile nella sezione [MAP] del file di progetto; se il valore non può essere trovato nella sezione [MAP], la Guida di Windows visualizza un messaggio di errore. Se non si conosce il valore del context number ma si conosce il valore del context string, è possibile utilizzare la seguente macro per visualizzare un determinato argomento:

```
JumpId ("FileName", "context_string")
```

Può capitare, inoltre, di non conoscere né il valore di context-string né il valore di context-number; in tal caso ci viene in aiuto la macro JumpKeyword ("Filename", "keyword") , la quale apre il file indicato da *Filename* e cerca tra i capitoli della guida il valore *keyword*, visualizzando il primo tra quelli trovati.

Si può utilizzare la macro JumpHelpOn() per richiamare "Uso della Guida; il file "Uso della Guida" è impostato per default a WINHELP.HLP, ma è possibile sostituirlo con un file di nostra creazione, utilizzando la macro SetHelp OnFile () , descritta nella tabella 1.

È possibile inoltre visualizzare delle finestre pop-up: per far ciò è necessario conoscere il context id oppure il context number dell'argomento desiderato; queste macro possono essere utili per spiegazioni aggiuntive (questo meccanismo è usato dalla stessa guida di Visual Basic). Le macro disponibili sono le seguenti:

```
PopUpContext ("Filename", context_number)
PopUpId ("Filename", "context_string")
```

La prima macro ha la funzione di visualizzare l'argomento identificato da *context_number* del file *Filename* in una finestra pop-up; la seconda è simile, ma che assume come valore la stringa *context_string* contenuta nella sezione [MAP].

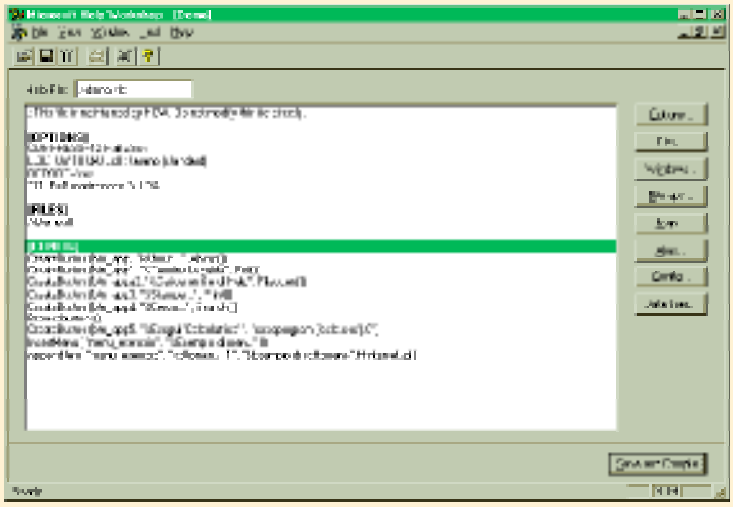


Figura 3

Il nuovo Help Compiler Workshop

UTILIZZO DI FUNZIONI IN DLL

L'Help Compiler permette di richiamare delle funzioni in DLL installate nel sistema, utilizzando la macro Register Routine ; questa macro registra una funzione contenuta in una DLL come una macro; le funzioni registrate possono essere richiamate da qualsiasi macro (hotspot o note a piè di pagina). È necessario comunque dichiarare la funzione nella sezione [CONFIG] del file di progetto utilizzando i seguenti parametri:

```
RegisterRoutine("DLLname",
    "function_name", "format_spec")
```

dove *DLLname* è il nome della DLL che contiene la funzione desiderata; se la DLL non esiste viene visualizzato un messaggio di errore e la macro è ignorata; *function_name* è il nome della funzione contenuta nella DLL che ci interessa utilizzare. Infine, *format_spec* è una stringa che specifica il formato dei parametri passati alla funzione: i caratteri nella stringa rappresentano dei tipi di caratteri del linguaggio C:

u	per unsigned short
U	per unsighed long
i	per short int
I	per long int
s	per string (near char *)
S	per string (far char *)
v	per void

A esempio, se volessimo aggiungere un effetto sonoro alla nostra guida eseguendo il file TADA.WAV dovremmo aggiungere la seguente riga nella sezione [CONFIG] del file progetto:

```
RegisterRoutine ("mmsystem",
    "sndPlaySound", "S")
```

In questo modo si richiama la funzione *sndPlaySound* della libreria MMSYSTEM .DLL, utilizzando come tipo di parametro una stringa. A questo punto sarà sufficiente richiamare la DLL da una macro nel modo seguente:

```
!sndPlaySound ("TADA.WAV", 0)
```

Questa macro può essere inserita così com'è in una nota a piè di pagina o come testo nascosto; può inoltre essere utilizzata da un pulsante o da un menu. Per creare un pulsante che esegua la macro *!sndPlaySound* e che abbia come testo "Esegui WAV", sarà necessario scrivere:

```
CREATEBUTTON ("btn_app", "&Esegui",
    "!sndPlaySound ('TADA.WAV', 0)")
```

Con questo secondo articolo termina la descrizione dell'Help Compiler 3.0; prossimamente ci rivedremo sulle pagine di VB JOURNAL per un resoconto delle novità introdotte dalla nuova versione 4.0 e dell' Help Compiler Workshop (vedi figura 3).

Roberto Scano ha conseguito il Diploma di Ragioniere e Perito Commerciale lo scorso anno. Ha esperienza di produzione e vendita software sia nel mercato shareware che commerciale, collabora spesso con riviste informatiche e da due anni si occupa di programmazione Windows in Visual Basic. Può essere raggiunto tramite telefono o fax al numero 041-5263540