```
    'Demo for Hiding/Showing Submenus and Command Items

'This example illustrates how to hide submenus and reshow them when de
sired. The menus are created in the normal way,
'and we keep track of handles, IDs and positions to refer to later.
'Terminology: Menus are basically lists of menu items. The menu bar at
 the top of the window is the top-level menu.
'Its menu items are normally themselves menus and are called submenus.
 The menu items listed in a submenu, such as
'the File menu, normally trigger some action, such as Save File, and a
re called command items. However, the File
'menu could contain a submenu that opens another list. Everything is a
 menu item. Each menu item is either itself a menu,
'or it is a command item. Every menu except the menu bar is also a sub
menu. We don't deal here with hiding
'the menu bar, so everything we hide/show will either be a submenu or
a command item. The distinction is important
'mainly because submenus are referred to by their handles (and normall
y have IDs of -1) and command items are
'referred to by their IDs (and have NULL handles).
'
'We create a File and Options menu, and two additional menus to trigge
r the hiding/showing of the entire Options
'menu, or the Save menu item in the File menu. Hiding/showing the Opti
ons menu illustrates how to deal with submenus.
'Hiding/showing the Save menu item illustrates how to deal with comman
d items.

    'declare major variables
    global hWindow  'Windows handle to the window
    global hMenuBar 'Windows handle to menu bar in window

    global hFileMenu, hOptionsMenu
'Windows handles to menus we want to mess with
    global menuSaveID     'ID of Save menu item
    global menuFilePosition, menuSavePosition,
 menuOptionsPosition
'Positions of these menu items in their menus, with first menu=0
    global showOptions, showSave
'Flags to indicate whether Options menu and Save menu item should be m
ade visible
    global optionsIsShowing, saveIsShowing
'Keeps track of what is actually showing now
    nomainwin

[CreateWindow]
```

```
    WindowWidth = 300
    WindowHeight = 300
    UpperLeftX = 200
    UpperLeftY = 200
        'Create File, Save and Options menus
    menu #handle, "File", "Open", [Open], "Save", [Save]
    menu #handle, "Options", "Calibrate", [Calibrate], "Reset", [
Reset]

'Create menu to hide/show entire Options menu with its command items
    menu #handle, "Options_Mode", "With Options",[
WithOptions], "Without Options", [WithoutOptions]

'Create menu to hide/show just the Save command item in the File menu
    menu #handle, "File_Mode", "With Save", [WithSave],
"Without Save", [WithoutSave]

    open "Test Window" for window as #handle
'Open the actual window with all menus showing


'Keep track of the position of the things that need to be modified. No
te that if we were possibly

'hiding more than one menu in the menu bar, the position of our target
 menu might change based on

'what else we had hidden, so we would have to keep track of that.
    menuOptionsPosition=1
'Options menu is second in menu bar, which is position 1 when indexed
from zero
    menuFilePosition=0
'File menu is first in menu bar, which is position 1 when indexed from
 zero
    menuSavePosition=1
'Save menu item is second in File menu, which is position 1 when index
ed from zero

    hWindow=hwnd(#handle)  'get window handle

'Get handle to the menu bar. We could put this in a subroutine, but it
 will only be used once.
    calldll #user32, "GetMenu",_
        hWindow as ulong,_ 'window handle
        hMenuBar as ulong 'returns handle of menubar
```

```
'Save handle to Options menu for later reference. Because it is a subm
enu (i.e. it opens another
        'menu of choices) it is referred to by its handle)
    hOptionsMenu=uSubMenuHandle(hMenuBar, menuOptionsPosition)
    hFileMenu=uSubMenuHandle(hMenuBar, menuFilePosition)

'Save ID of Save menu item. Because it is a command item (i.e. it trig
gers a command, rather than
        'opening another menu), it is referred to by an ID number)
    menuSaveID=uMenuItemID(hFileMenu, menuSavePosition)

    optionsIsShowing=1 : saveIsShowing=1
'Everything is showing at the moment
    showSave=1 : showOptions=0
'Tells ConformMenusToMode to show Save and hide Options

    call ConformMenusToMode
'Hide/show whatever is indicated by showSave and showOptions
    print #handle, "trapclose [quit]"
'goto [quit] if window is closed

    wait    'Wait for user action

[quit]
    close #handle
    end

'These are the actions for the File and Options menus. Here we just wa
it.
[Open]
    wait
[Save]
    wait
[Calibrate]
    wait
[Reset]
    wait


'The following respond to clicks in the Options Mode and Save Mode men
us.
[WithOptions]
    showOptions=1 : call ConformMenusToMode
    wait
[WithoutOptions]
```

```
    showOptions=0 : call ConformMenusToMode
    wait
[WithSave]
    showSave=1 : call ConformMenusToMode
    wait
[WithoutSave]
    showSave=0 : call ConformMenusToMode
    wait

'ConformMenusToMode shows/hides whatever is indicated by showOptions a
nd showSave
'showOptions and showSave could be made arguments here instead of glob
al variables.
sub ConformMenusToMode

'Here we hide everything that might ever need to be hidden, and then s
how what we want. In this

'simple example, we could first check to see that what is now shown is
 supposed to be shown, but

'for a more involved case it is better to first hide everything that m
ight need to be hidden.


'We created two different subroutines for hiding. One is for command i
tems like Save, which

'actually trigger action and are referred to by an ID. One is for subm
enus like Options, which simply
        'open another list of choices.
    if saveIsShowing then menuOK=uHideCommandItem(hFileMenu,
menuSaveID)   'Hide Save menu item, identified by the handle of

  'the menu it is in and its own ID


'Hide Options menu item, identified by the handle of menu it is in and
 its relative postion.

'This hides the word "Options" in the menu bar and also the Calibrate
and Reset command items

'that are listed ini the Options menu. Because uHideSubMenu uses the D
LL RemoveMenu, rather than DeleteMenu,
'the full Options menu structure will continue to exist in memory so w
e can show it later.
    if optionsIsShowing then menuOK=uHideSubMenu(hMenuBar,
 menuOptionsPosition)


'Now both Options and Save are hidden. If we want to show one, we do i
t here.

'For variety, we take a different approach here. For hiding, we create
d separate subroutines

'for command items and menus. Here, we use a single subroutine, with t
wo different arguments, one for

'the ID and one for the handle. When showing a command item, we use th
e ID and dummy handle. When showing
        'a submenu, we use the handle and a dummy ID.
    if showSave then
        menuOK=uShowMenuItem(hFileMenu, _
'Handle to the menu that contains the Save menu item
                menuSaveID, _
'ID of the Save menu item. We use this because Save is a command item
                NULL, _
'Dummy handle for Save menu item. We use this because command items do
n't have handles.
                "Save", _
'Caption to use for inserted menu item. We aren't changing the caption
.
                menuSavePosition)
'Position in File menu to insert Save menu item.
        saveIsShowing=1
    else
        saveIsShowing=0
    end if

    if showOptions then
        menuOK=uShowMenuItem(hMenuBar, _
'Handle to the menu that contains the Save menu item
                -1, _
'Dummy ID of Options menu. IDs are for command items, which this is no
t.
                hOptionsMenu, _
'handle to Options menu, which still exists in memory
                "Options", _
'Caption to use for inserted menu. We aren't changing the caption.
```

```
                menuOptionsPosition)
'Position in menu bar to insert Options menu
        optionsIsShowing=1
    else
        optionsIsShowing=0
    end if


'The above only affected memory structures. We need to actually draw t
he new menu.

'This has to be done even if the window is not currently visible.

'We specify the handle of the window whose menu bar is to be redrawn.
    call uDrawMenu hWindow
end sub

'----------------------------------------------------------------------
----------------------
'-----The remaining subroutines are wrappers for calling Windows API f
unctions.-------------
'----------------------------------------------------------------------
----------------------
function uHideCommandItem(hParent, itemID)
    'Delete the specified command item. It can later be re-inserted

'hParent is the submenu containing the item. "Parent" may not be Windo
ws terminology.
    'itemID is the ID of the item to hide by removing it
    calldll #user32, "RemoveMenu",_
        hParent as ulong,_ 'handle of submenu
        itemID as ulong,_ 'menu item ID
        _MF_BYCOMMAND as long,_
'says we are specifying the target item by its ID, not its position
        result as long
'nonzero=success We return this, but it probably won't be used.
    uHideCommandItem=result
end function

function uHideSubMenu(hParent, position)
    'Delete the specified submenu. It can later be re-inserted

'hParent is the submenu containing the item. "Parent" may not be Windo
ws terminology.
    'postion is the position of the item to hide by removing it
    calldll #user32, "RemoveMenu",_
```

```
        hParent as ulong,_
'handle of menu listing the one we want to hide
        position as ulong,_ 'position of menu we want to hide
        _MF_BYPOSITION as long,_
'Says we are specifying the target menu by its position
        result as long
'nonzero=success We return this, but it probably won't be used.
    uHideSubMenu=result
end function

function uShowMenuItem(hParent, itemID, hMenu, caption$, precedeNum)
    'Insert a menu item that was previously deleted
    'hParent is the submenu in which it will be inserted

'itemID is the item ID of the item to insert, if it is a command item;
 otherwise -1

'hMenu is the handle of the submenu to insert, if the item is a submen
u; otherwise NULL
    'caption$ is the name of the item being inserted;

'The item will be inserted prior to the item in position precedeNum (0
...)--That is,
    'once it is inserted it will occupy position precedeNum.
    'the number of existing items.

'An alternative subroutine could be written to locate the insertion

'point by the ID of the item to be preceded, rather than by position.


'The DLL uses the following structure to contain info about the item t
o be shown
    struct MENUITEMINFO,cbSize as ulong,fMask as ulong, _
        fType as ulong,fState as ulong,wID as ulong, _
        hSubMenu as ulong,hbmpChecked as ulong,
hbmpUnchecked as ulong, _
        dwItemData as ulong,dwTypeData$ as ptr,cch as ulong
'Note dwTypeData$ is ptr because we have text caption
    MENUITEMINFO.cbSize.struct = len(MENUITEMINFO.struct)
'length of this structure
    MENUITEMINFO.fMask.struct = _MIIM_ID or
 _MIIM_SUBMENU or _MIIM_TYPE
'Says to process ID, submenu handle and type
    MENUITEMINFO.wID.struct = itemID
'ID of command item, or any value for submenu
```

```
    MENUITEMINFO.fType.struct = _MFT_STRING 'type of caption is text
    MENUITEMINFO.hSubMenu.struct =hMenu
'handle to submenu being inserted, or NULL if not a submenu
    MENUITEMINFO.dwTypeData$.struct = caption$
'caption for menu. This is represented in the struct as a pointer to t
he text
    MENUITEMINFO.cch.struct = len(caption$) 'length of new caption

'we don't mess here with the struct members fState, hbmChecked, hbmpUn
checked or dwItemData.

        'Actual DLL call
    calldll #user32, "InsertMenuItemA",_
        hParent as ulong,_
'handle of menu into which we are inserting. "Parent" may not be Windo
ws terminology.
        precedeNum as ulong,_ 'pos of item that new item is to precede
        _MF_BYPOSITION as long,_
'Says to find locate the insertion point by position
        MENUITEMINFO as struct,_ 'struct with item info
        result as long
'nonzero=success We return this, but it probably won't be used.
    uShowMenuItem=result
end function

sub uDrawMenu hWind  'Redraw menu after making modifications
    calldll #user32, "DrawMenuBar",_
        hWind as ulong,_ 'window handle
        result as long 'nonzero=success We don't bother to return this
end sub

function uSubMenuHandle(hParent, subPosition)
'Return handle of submenu in specified position

'hParent is handle of the menu bar, or whatever menu contains the subm
enu
    'subPosition is position (0...) of the desired submenu
    calldll #user32, "GetSubMenu",_
        hParent as ulong,_ 'menu handle
        subPosition as long,_
'0-indexed pos of submenu whose handle is sought
        hSub as ulong 'returns submenu handle
    uSubMenuHandle=hSub
end function

function uMenuItemID(hSub, itemPosition)
```

```
'Return ID of item in submenu in menubar

'This returns -1 if the specified menu item is not a command item, so
    'it is pointless to use it for submenus.
    calldll #user32, "GetMenuItemID", _
        hSub as ulong, _ 'handle of the submenu
        itemPosition as long, _ 'position of the menu item
        menuID as ulong 'the handle (or ID) of the menu item
    uMenuItemID=menuID
end function
```