MessageBoxExA

by Alyce Watson

adapted from *APIs for Liberty BASIC*
http://www.lulu.com/content/611431

Table of Contents

The MessageBoxExA function creates, displays, and operates a messagebox window. This is similar to Liberty BASIC's NOTICE and CONFIRM statements, but it allows more flexibility. You can select the icon and the desired buttons for this dialog, as well as the desired modality. This is handy if you want to give your users a choice of "yes, no or cancel", "abort, retry, ignore" etc.

The messagebox displays the text and title that you specify. It also displays a combination of the predefined icons and push buttons described below.

The function also has an argument for the handle of your program's window as the owner of the messagebox. The handle argument can be null. (A null value is equal to 0.)

Parameters

hWnd

Identifies the owner window of the message box to be created. If this parameter is NULL, the message box has no owner window. If you create a message box while a dialog box is present, use the handle of the dialog box as the hWnd parameter. The hWnd parameter should not identify a child window.

To retrieve a Liberty BASIC window's handle, use HWND(#handle) like this:

```
open "A window" for window as #main
handle = hwnd(#main)
```

lpCaption$

Points to a null-terminated string containing the message to be displayed.

lpszTitle$

Points to a null-terminated string used for the dialog box title. If this parameter is NULL, the default title Error is used.

uType

Specifies a set of bit flags that determine the contents and behavior of the dialog box. This parameter can be a combination of flags from the following groups of flags.

COMBINING CONSTANTS

To set multiple bit flags for the "uType" paramenter, combine the values with the Liberty BASIC **OR** operator.

BUTTONS

Specify one of the following flags to indicate the buttons contained in the message box:

_MB_ABORTRETRYIGNORE The message box contains three push buttons: Abort, Retry, and Ignore.
_MB_OK The message box contains one push button: OK. This is the default.
_MB_OKCANCEL The message box contains two push buttons: OK and Cancel.
_MB_RETRYCANCEL The message box contains two push buttons: Retry and Cancel.

_MB_YESNO The message box contains two push buttons: Yes and No.
_MB_YESNOCANCEL The message box contains three push buttons: Yes, No, and Cancel.

ICONS

Specify one of the following flags to display an icon in the message box:

_MB_ICONEXCLAMATION, or _MB_ICONWARNING An exclamation-point icon appears in the message box.
_MB_ICONINFORMATION, or _MB_ICONASTERISK An icon consisting of a lowercase letter i in a circle appears in the message box.
_MB_ICONQUESTION A question-mark icon appears in the message box.
_MB_ICONSTOP, _MB_ICONERROR,_MB_ICONHAND A stop-sign icon appears in the message box.

DEFAULT BUTTON

The "default button" is the one that is activated if the user presses the "Enter" key on his keyboard, rather than clicking a button in the messagebox with his mouse.

Specify one of the following flags to indicate the default button:

_MB_DEFBUTTON1 The first button is the default button. MB_DEFBUTTON1 is the default unless a different button is specified.
_MB_DEFBUTTON2 The second button is the default button.
_MB_DEFBUTTON3 The third button is the default button.
_MB_DEFBUTTON4 The fourth button is the default button.

MODALITY

Specify one of the following flags to indicate the modality of the dialog box:

_MB_APPLMODAL The user must respond to the message box before continuing work in the window identified by the hWnd parameter. However, the user can move to the windows of other applications and work in those windows. Depending on the hierarchy of windows in the application, the user may be able to move to other windows within the application. All child windows of the parent of the message box are automatically disabled, but popup windows are not.MB_APPLMODAL is the default if neither _MB_SYSTEMMODAL nor _MB_TASKMODAL is specified.
_MB_SYSTEMMODAL Same as MB_APPLMODAL except that the message box has the WS_EX_TOPMOST style. Use system-modal message boxes to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory). This flag has no

effect on the user's ability to interact with windows other than those associated with hWnd.
_MB_TASKMODAL Same as MB_APPLMODAL except that all the top-level windows belonging to the current task are disabled if the hWnd parameter is NULL. Use this flag when the calling application or library does not have a window handle available but still needs to prevent input to other windows in the current application without suspending other applications.

OTHER OPTIONS

In addition, you can specify the following flags:

_MB_RIGHT The text is right-justified.
_MB_SETFOREGROUND The message box becomes the foreground window. Internally, Windows calls the SetForegroundWindow function for the message box.
_MB_TOPMOST The message box is created with the WS_EX_TOPMOST window style.

wLanguageId
Specifies the language in which to display the text contained in the predefined push buttons. We will not specify a different language in our demo.

Return Values

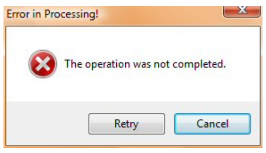
If the function succeeds, the return value is a nonzero menu-item value returned by the dialog box.

_IDABORT Abort button was selected.
_IDCANCEL Cancel button was selected.
_IDIGNORE Ignore button was selected.
_IDNO No button was selected.
_IDOK OK button was selected.
_IDRETRY Retry button was selected.
_IDYES Yes button was selected.

If a message box has a Cancel button, the function returns the IDCANCEL value when either the ESC key or Cancel button is pressed. If the message box has no Cancel button, pressing the ESC key has no effect.

If the function fails, the return value is zero.

Here is a demonstration program. When run, it creates a messagebox that looks like this on Windows Vista:


Error in Processing!

The operation was not completed.

Why not experiment with different buttons and icons and see if you can recreate Liberty BASIC's NOTICE and CONFIRM messageboxes?

```
hWnd = 0
lpCaption$ = "The operation was not completed."
lpszTitle$ = "Error in Processing!"
utype = _MB_ICONSTOP or _MB_RETRYCANCEL or _MB_DEFBUTTON2
wLanguageId = 0

calldll #user32, "MessageBoxExA",_
hWnd as ulong,_ 'window handle
lpCaption$ as ptr,_ 'desired message text
lpszTitle$ as ptr,_ 'desired titlebar caption
utype as long,_ 'flag for icon and buttons
wLanguageId as word,_ 'language identifier
result as long 'returns action code

if result = _IDCANCEL then print, "Cancel button was selected."
if result = _IDRETRY then print, "Retry button was selected."
```

For another demonstration program and for code to use the Borland messagebox DLL, see this page: http://alycesrestaurant.com/messageboxes.htm

Windows Constants

Liberty BASIC knows the values of some Windows constant. In those cases, you can use the Windows constant name, preceded by an underscore character.

If you want to know the numeric value of a Windows constant, write some code to print it out, as below. If Liberty BASIC halts with an "undefined constant" error, you'll have to search the DLL documentation to find the value yourself.

```
print _IDABORT
print _IDCANCEL
print _IDIGNORE
print _IDNO
print _IDOK
print _IDRETRY
print _IDYES
print _MB_HELP
print _MB_RIGHT
print _MB_SETFOREGROUND
print _MB_TOPMOST
print _MB_APPLMODAL
print _MB_SYSTEMMODAL
print _MB_TASKMODAL
print _MB_ABORTRETRYIGNORE
print _MB_OK
print _MB_OKCANCEL
print _MB_RETRYCANCEL
print _MB_YESNO
print _MB_YESNOCANCEL
print _MB_ICONEXCLAMATION
print _MB_ICONWARNING
print _MB_ICONINFORMATION
print _MB_ICONASTERISK
print _MB_ICONQUESTION
print _MB_ICONSTOP
print _MB_ICONERROR
print _MB_ICONHAND
print _MB_DEFBUTTON1
print _MB_DEFBUTTON2
print _MB_DEFBUTTON3
print _MB_DEFBUTTON4
```

More on Windows constants can be found here:
ABCs of APIs 8