DOWNLOAD PDF THE DIALOG PROCEDURE DLGPROC

Chapter 1 : How to close a dialog without closing the whole application?

If the dialog box procedure returns FALSE, the dialog manager performs the default dialog operation in response to the message. If the dialog box procedure processes a message that requires a specific return value, the dialog box procedure should set the desired return value by calling SetWindowLong (hwndDlg, DWL_MSGRESULT, lResult) immediately before returning TRUE.

Each method consists of a simple kernel of an idea; the rest is just scaffolding to make the kernel work. The first way uses a recursive call from the dialog procedure back into DefDlgProc to trigger the default behavior. This technique requires that you have a flag that lets you detect and therefore break the recursion. The kernel is to "subvert the recursive call". DefDlgProc calls your dialog procedure to see what you want to do. When you want to do the default action, just call DefDlgProc recursively. The inner DefDlgProc will call your dialog procedure to see if you want to override the default action. The recursive DefDlgProc will then perform the default action and return its result. Now you have the result of the default action, and you can modify it or augment it before returning that as the result for the dialog box procedure, back to the outer DefDlgProc which returns that value back as the final message result. I call it a Wndproc-Like Dialog: The default implementation in the base class uses the DefDlgProcEx macro from windowsx. In fact, my first version did exactly that. After short-circuiting messages that arrive before the dialog box has initialized, it uses the CheckDlgRecursion macro, also from windowsx. This macro checks the recursion flag; if set, then it resets the flag and just returns FALSE immediately. This is what stops the recursion. Otherwise, it calls the WLDlgProc method which has probably been overriden in a derived class , then sets the dialog procedure return value and returns. The SetDlgMsgResult macro also comes from windowsx. Well, unless the message is one of the special exceptions, in which case it returns the value directly. Note to bit developers: There is a bug in this macro as currently written. The last method is DoModal, which initializes the recursion flag and kicks off the dialog box. This is quite handy since it saves the reader the trouble of figure out "So which level in the class hierarchy are we forwarding this call to? WLDlgProc without the underscores. Can it be global?

Chapter 2 : Allow No DlgProc? - PowerBASIC Peer Support Community

Creates a modal dialog box from a dialog box template resource. Before displaying the dialog box, the function passes an application-defined value to the dialog box procedure as the lParam parameter of the WM_INITDIALOG message.

Multi-line input box Introduction One day, I was writing a small Windows tool, and wanted to get some input from the user. Since my application was not an MFC, nor a console application, to my knowledge, there was no simple way to get the input from the user. Since I wanted to keep my application slim and self-contained, I decided to investigate on how to use dialogs without using resources and without introducing too much of code. Thus the Win32InputBox library was born. In this article, I will illustrate how to create and use dialog boxes without creating dialog resources or using MFC. Nonetheless, I will be giving a simple introduction for beginners. Introduction to dialog boxes - the classical way Currently, to create a dialog box driven application, you have many choices. MFC dialog based application can be generated by the wizard. Plain Win32 API - without using the dialog designer, instead, using the code to create all the controls and the dialog window. We start by designing our dialog with the editor. We create controls buttons, textboxes, We assign IDs to the controls. We write our WindowProc , which will handle all the events related to our dialog. We call the appropriate dialog creation method CreateDialog family, or DialogBox function family. If you are not using the dialog designer, you will have to create the controls in the code by calling the CreateWindow function. As for MFC dialog based applications, the concept is similar, however everything is wrapped into neat classes. So, all you have to do is: Design the dialog using the editor. Bind your subclassed class to the desired dialog ID. Introduction to dialog templates What is a dialog template? It is a structure that defines the styles and dimensions of a given dialog. There is an extended version of this structure that renders the latter obsolete. When you compile your application, the resources get compiled separately using the RC. EXE resource compiler tool, which will produce the. RES binary file out of the. RC text file , and finally everything will be linked together to produce one executable module. However, I would like you to note the following: The numbers which denote the dimensions and positions. Those IDS are defined in the "resource. RES file looks like, I have included a small hex dump of this compiled structure here: RC file icons, string table, etc Since the resource file holds a number of resource items, all defined by certain IDs and resource types, we need a way to select the given resource and use it. This small code will illustrate how to create a dialog whose ID is defined in resource. LoadResource hModule, hrsrc ;:: This is the pointer to the compiled dialog template as discussed earlier. A simple dialog procedure which we can write is: After extracting the dialog template, we can employ it in our own application. EXE template in our own simple application. Extract the template from notepad. Now that we have the "n. Next, we will see how practical it is to use the dialog templates to create small self-contained functions. CWin32InputBox class overview Now we can talk about the construction of the CWin32Inputbox class, after having introduced all the concepts needed for this task. In a nutshell, CWin32InputBox:: Call DialogBoxIndirectParam to show the dialog. Do some actions based on the return value of the modal dialog. The class exports the two static methods: So to use this class in your project, all you have to do is simply add the "Win32InputBox. No need for resources or anything. Points of interest It was fun learning about dialog templates and more fun to write the reusable CWin32InputBox class. Hope you enjoyed and learned from this article. License This article has no explicit license attached to it but may contain usage terms in the article text or the download files themselves. If in doubt please contact the author via the discussion board below. A list of licenses authors might use can be found here Share.

Chapter 3 : Windows API Guide: SetWindowLong Function

Typically, the dialog box procedure should return TRUE if it processed the message, and FALSE if it did not. If the dialog box procedure returns FALSE, the dialog manager performs the default dialog operation in response to the message.

CDialogImpl Dialogs represent a declarative style of user interface development. Whereas normal windows provide all kinds of flexibility you can put anything you want in the client area of a window , dialogs are more static. Actually, dialogs are just windows whose layout has been predetermined. The built-in dialog box window class knows how to interpret dialog box resources to create and manage the child windows that make up a dialog box. To show a dialog box modallythat is, while the dialog is visible, the parent is inaccessibleWindows provides the DialogBoxParam [6] function: To show the dialog box modelesslythat is, the parent window is still accessible while the dialog is showingthe CreateDialogParam [7] function is used instead: The return value from Create-DialogParam represents the HWND of the new dialog box window, which will live until the dialog box window is destroyed. Regardless of how a dialog is shown, however, developing a dialog is the same. First, you lay out a dialog resource using your favorite resource editor. Second, you develop a dialog box procedure DlgProc. This is the same kind of grunt work we had to do when we wanted to show a window that is, register a window class, develop a WndProc, and create the window. First, notice the use of the thunk. In addition to all the tricks that WindowProc performs see the section " The Window Procedure ," earlier in this chapter , the static member function CDialogImpl:: For some dialog messages, the DlgProc must return the result of the message. And although I can never remember which is which, ATL can. Also notice the DefWindowProc member function. You might wonder how far the inheritance hierarchy goes for CDialogImpl. Refer again to Figure Notice that CDialogImpl ultimately derives from CWindow, so all the wrappers and helpers that are available for windows are also available to dialogs. Before we get too far away from CDialogImpl, notice where the dialog resource identifier comes from. The deriving class is required to provide a numeric symbol called IDD indicating the resource identifier. Using a CDialogImpl-derived class is a matter of creating an instance of the class and calling either DoModal or Create: This reduces the definition of the CAboutBox class to the following type definition: When you build a dialog using the Visual Studio dialog editor and you add a combo box, the contents of that combo box are not stored in the regular Dialog resource. The normal dialog APIs completely ignore this initialization data. In fact, most are downright complicated. This complication is mostly due to two reasons: Exchanging data with a modal dialog typically goes like this: The application creates an instance of a CDialogImpl-derived class. The application calls DoModal. The dialog handles the OK button by validating that data held by child controls. If the data is not valid, the dialog complains to the users and makes them keep trying until either they get it right or they get frustrated and click the Cancel button. If the dialog is to be shown modelessly, the interaction between the application and the dialog is a little different, but the relationship between the dialog and its child controls is the same. A modeless dialog sequence goes like this differences from the modal case are shown in italics: The applications calls Create. The dialog handles the Apply button by validating that data held by child controls. When the application is notified, it copies the data from the dialog data members over its own copy. ATL has no such support, but it turns out to be quite easy to build yourself. For example, to beef up our standalone windows application sample, imagine a dialog that allows us to modify the display string, as shown in Figure A dialog that needs to manage data exchange and validation The CDialogImpl-based class looks like this: OnInitDialog copies the data from the data member into the child edit control. Likewise, OnOK copies the data from the child edit control back to the data member and ends the dialog if the data is valid. I decided that a zero-length string would be too boring, so I made it invalid. I can do even better in the data validation area with this example. If the string ever gets to zero, disabling the OK button would make it impossible for the user to attempt to commit the data at all, making the complaint dialog unnecessary. The following updated sample shows this technique:

Chapter 4 : CreateDialogParam function (Windows)

In the Dialog class I have a member function defined as INT_PTR CALLBACK Dialog::cb_proc(HWND,UINT,WPARAM,LPARAM). Now, I know that windows must have a global function as a callback procedure. So I made a std::map DlgProcs map to associate the dialogs window handle with its Dialog class pointer.

Next page Dialogs represent a declarative style of user interface development. Whereas normal windows provide all kinds of flexibility you can put anything you want in the client area of a window , dialogs are more static. Actually, dialogs are just windows whose layout has been predetermined. The built-in dialog box window class knows how to interpret dialog box resources to create and manage the child windows that make up a dialog box. To show a dialog box modallythat is, while the dialog is visible, the parent is inaccessibleWindows provides the DialogBoxParam[6] function: To show the dialog box modelesslythat is, the parent window is still accessible while the dialog is showingthe CreateDialogParam[7] function is used instead: The return value from Create-DialogParam represents the HWND of the new dialog box window, which will live until the dialog box window is destroyed. Regardless of how a dialog is shown, however, developing a dialog is the same. First, you lay out a dialog resource using your favorite resource editor. Second, you develop a dialog box procedure DlgProc. This is the same kind of grunt work we had to do when we wanted to show a window that is, register a window class, develop a WndProc, and create the window. First, notice the use of the thunk. In addition to all the tricks that WindowProc performs see the section "The Window Procedure," earlier in this chapter , the static member function CDialogImpl:: For some dialog messages, the DlgProc must return the result of the message. And although I can never remember which is which, ATL can. Also notice the DefWindowProc member function. You might wonder how far the inheritance hierarchy goes for CDialogImpl. Refer again to Figure Notice that CDialogImpl ultimately derives from CWindow, so all the wrappers and helpers that are available for windows are also available to dialogs. Before we get too far away from CDialogImpl, notice where the dialog resource identifier comes from. The deriving class is required to provide a numeric symbol called IDD indicating the resource identifier. Using a CDialogImpl-derived class is a matter of creating an instance of the class and calling either DoModal or Create: This reduces the definition of the CAboutBox class to the following type definition: When you build a dialog using the Visual Studio dialog editor and you add a combo box, the contents of that combo box are not stored in the regular Dialog resource. The normal dialog APIs completely ignore this initialization data. In fact, most are downright complicated. This complication is mostly due to two reasons: Exchanging data with a modal dialog typically goes like this: The application creates an instance of a CDialogImpl-derived class. The application calls DoModal. The dialog handles the OK button by validating that data held by child controls. If the data is not valid, the dialog complains to the users and makes them keep trying until either they get it right or they get frustrated and click the Cancel button. If the dialog is to be shown modelessly, the interaction between the application and the dialog is a little different, but the relationship between the dialog and its child controls is the same. A modeless dialog sequence goes like this differences from the modal case are shown in italics: The applications calls Create. The dialog handles the Apply button by validating that data held by child controls. When the application is notified, it copies the data from the dialog data members over its own copy. ATL has no such support, but it turns out to be quite easy to build yourself. For example, to beef up our standalone windows application sample, imagine a dialog that allows us to modify the display string, as shown in Figure A dialog that needs to manage data exchange and validation The CDialogImpl-based class looks like this: OnInitDialog copies the data from the data member into the child edit control. Likewise, OnOK copies the data from the child edit control back to the data member and ends the dialog if the data is valid. I decided that a zero-length string would be too boring, so I made it invalid. I can do even better in the data validation area with this example. If the string ever gets to zero, disabling the OK button would make it impossible for the user to attempt to commit the data at all, making the complaint dialog unnecessary. The following updated sample shows this technique:

Chapter 5 : KFWin - Fortan Windows Programming

Window proc vs. Dialog proc? What is the difference between a window procedure and a dialog procedure? I tried subclassing a dialog and it didn't work when I used DWL_DLGPROC but worked OK when I used GWL_WNDPROC.

Use -1 to hide all pages. It is list of page indices separated by spaces. For example, "0 0 1 2 2 -1" shows page 0 when index is 0 or 1, page 1 when index is 2, page 2 when index is 3 or 4, and none when index is 5. Use parentheses to show several pages simultaneously. For example, "0 1 3 2" shows pages 1 and 3 when index is 1. You can also use the same string in dialog editor Options. Then mapping will be applied at design time. A macro can contain more than 1 dialog. Use sub-functions for it. DlgProc2 0 hwndOwner ret In this example there are 2 dialogs. One in the parent function and one in sub-function Dialog2. When opening the Dialog Editor, you can choose which dialog to edit. If a sub-function or parent function contains multiple dialogs, the Dialog Editor uses the first dialog in it the first dialog definition, ShowDialog statement, variable declarations, and the first found dialog procedure below it. To enable it for whole program, check the checkbox in Options. If QM is running in Unicode mode, most controls work well with any text regardless of dialog Unicode mode. When Unicode is enabled for a dialog, its dialog procedure receives Unicode versions of some messages, particularly those that can pass or query text. Then the text is in Unicode UTF format. You can use str. Common controls tree view, list view, etc have their own Unicode mode, which is enabled in Unicode dialogs, and disabled in ANSI dialogs. Create "Courier New" 14 1 f. SetDialogFont hDlg "" f. Create "Comic Sans MS" 7 2 f2. SetDialogFontColor hDlg 0x "6" note: It can be the name of a macro or item of other type that contains menu definition. It also can be menu definition itself. To create menu definitions, use the Menu Editor. Hyphen is used for separators. A line can optionally begin with any number of spaces and tabs. Syntax used for menu items: Use ampersand to underline characters. Can be used escape sequences not in Menu Editor. Should be 0 to 0xFFFF. Use bigger values e. Normally it is omitted or 0. The hotkey string is appended to the menu item label automatically, unless label contains a tab character. Other info In exe also can be used menus from resources. The menu argument of ShowDialog must be semicolon followed by resource id, e. If the resource contains accelerator table with the same id, the table is used for accelerators, although accelerator text is not added must be included in label text, after tab. Menu bars and accelerators can be used with modal and modeless dialogs, but not with child dialogs. Menu definitions also can be used to show popup menus. Use ShowMenu or MenuPopup. Common controls, other controls The Dialog Editor and dialog functions support controls of the following classes: Working with controls of these classes is easy. You can initialize them before you call ShowDialog, and get data text, checked, etc from them when dialog is closed. Dialogs can contain controls of other classes too. However, only programmers can use them. The dialog must be a "smart" dialog, otherwise the controls will be useless. To interact with these controls, are used various messages with SendMessage. Messages, styles and other information can be found in MSDN library. ActiveX controls Controls that you usually use in dialogs are Windows controls. Also you can use ActiveX controls. ActiveX controls are COM objects that provide graphical user interface. They are used differently than Windows controls. To work with Windows controls, are used messages. To work with ActiveX controls, are used functions that they provide. To insert an ActiveX control, click "ActiveX controls Select the control you need and click Add Control To Dialog. ActiveX controls are defined in type libraries , as COM classes. To use an ActiveX control in a dialog, at first declare the type library where the control class is defined. To insert type library declarations, use the COM Libraries dialog. When adding a control to a dialog, QM prompts to insert type library declaration if it still does not exist in current macro. Several type libraries, including SHDocVw web browser control are already declared. When you have an ActiveX control created, you usually need some way to communicate with it. Then you can call functions with the variable to manipulate the control. The variable must be local not global or thread. There is a sample dialog with web browser control in the forum. You cannot use str. If the text is URL or path of a html, image or other file, web browser control opens that web page or file. Does not wait until finished loading. Sets silent mode to avoid script error messages. If the text is "", loads "about: ActiveX controls can be used in other windows non-dialog too. Class name and text must be like in dialog definitions: QM supports run-time licensing of ActiveX controls. It allows you to add purchased ActiveX controls to macros and exes and distribute these macros and exes. With attached run-time license key, the control can be used on any computer. If you distribute macro not exe , you should encrypt the macro that contains the dialog definition to prevent others to reuse the run-time license key in their macros. Even if the macro is not encrypted, the key can only be used in Quick Macros, because only QM can decrypt the key. Also, others will not be able to create exes with the key. Note that some controls may use different ways to verify license. For example, you may have to call certain function and pass the license key. This is because they are designed for full-featured containers, such as Visual Basic forms, or only for certain programs. At design time, you can only move and resize ActiveX controls or control placeholders. Properties can be changed only at run time. Such components usually are in dll files. Controls usually are in ocx or dll files. It is possible to use some ActiveX controls even if the COM component is not registered on that computer. If the file not found, tries to create as registered.

Chapter 6 : Dialog Box Programming Considerations | Microsoft Docs

The procedure should close the dialog box by using the EndDialog function for modal dialog boxes and the DestroyWindow function for modeless dialog boxes. The system also sends WM_COMMAND messages to the dialog box procedure if the dialog box has a menu, such as the window menu, and the user clicks a menu item.

The predefined window procedure passes all other messages to DefWindowProc for default processing. Dialog Box Keyboard Interface The system provides a special keyboard interface for dialog boxes that carries out special processing for several keys. The interface generates messages that correspond to certain buttons in the dialog box or changes the input focus from one control to another. Following are the keys used in this interface and their respective actions. DOWN Moves the input focus to the next control in the group. LEFT Moves the input focus to the previous control in the group. UP Moves the input focus to the previous control in the group. The system automatically provides the keyboard interface for all modal dialog boxes. It does not provide the interface for modeless dialog boxes unless the application calls the IsDialogMessage function to filter messages in its main message loop. This means that the application must pass the message to IsDialogMessage immediately after retrieving the message from the message queue. The function processes the messages if it is for the dialog box and returns a nonzero value to indicate that the message has been processed and must not be passed to the TranslateMessage or DispatchMessage function. Because the dialog box keyboard interface uses direction keys to move between controls in a dialog box, an application cannot use these keys to scroll the contents of any modal dialog box or any modeless dialog box for which IsDialogMessage is called. When a dialog box has scroll bars, the application must provide an alternate keyboard interface for the scroll bars. Note that the mouse interface for scrolling is available when the system includes a mouse. The search starts with the control currently having the input focus and proceeds in the order in which the controls were createdâ€”that is, the order in which they are defined in the dialog box template. When the system locates a control having the required characteristics, the system moves the input focus to it. As long as the current control with the input focus does not process these keys and the next or previous control is not a static control, the system continues to move the input focus through all controls in the dialog box as the user continues to press the direction keys. The style marks the beginning of a group of controls. If a control in the group has the input focus when the user begins pressing direction keys, the focus remains in the group. All controls in the group must be contiguousâ€”that is, they must have been created consecutively with no intervening controls. When the user presses a direction key, the system first determines whether the current control having the input focus processes the direction keys. Otherwise, the system uses the GetNextDlgGroupItem function to determine the next control in the group. GetNextDlgGroupItem searches controls in the order or reverse order they were created. The system then moves the input focus to control if it is not a static control. The system automatically moves the style when the user moves between controls in the group. This ensures that the input focus will always be on the most recently selected control when the user moves to the group using the TAB key. Mnemonics A mnemonic is a selected letter or digit in the label of a button or in the text of a static control. The system moves the input focus to the control associated with the mnemonic whenever the user either presses the key that corresponds to the mnemonic or presses this key and the ALT key in combination. Mnemonics provide a quick way for the user to move to a specified control by using the keyboard. In most cases, the null-terminated string provided with the control in the dialog box template contains the ampersand. Only one mnemonic can be specified for each control. Although it is recommended, mnemonics in a dialog box need not be unique. When the user presses a letter or digit key, the system first determines whether the current control having the input focus processes the key. Otherwise, it searches for a control whose mnemonic matches the specified letter or digit. It continues to search until it locates a control or has examined all controls. If the system locates some other control that has a matching mnemonic, it moves the input focus to that control. Dialog Box Settings The dialog box settings are the current selections and values for the controls in the dialog box. The dialog box procedure is responsible for initializing the controls to these settings when creating the dialog box. It is also responsible for retrieving the current settings from the controls before destroying the dialog box. The methods used to initialize and retrieve settings depend on the type of control. For more information, see the following topics:

Chapter 7 : proplem with messagebox - C++ Forum

But if you are doing this from a dialog procedure, you have to do this in two steps: SetWindowLongPtr(hdlg, DWLP_MSGRESULT, TRUE); return TRUE; The second line sets the return value for the dialog procedure, which tells DefDlgProc that the message has been handled and default handling should be suppressed.

Chapter 8 : DialogProc callback function (Windows)

In WinMain, you are using DialogBox, which means you are creating a modal dialog box. You specify DlgProc for the modal dialog. For the WM_INITDIALOG message for the main dialog you call DialogBox again, this time for the tutorial dialog. You specify the same dialog proc (DlgProc) for the tutorial dialog. I am nearly certain that would be a problem.

Chapter 9 : Dialog Subclassing in 64 bit applications

As far as I know, you aren't supposed to change the dialog procedure (the thingo returned by DWL_DLGPROC) in run-time. You can change the window procedure (the thingo returned by GWL_WNDPROC). Maybe Win32 lets you do that, but -- I think correctly -- Win64 won't.