CHAPTER 11: DIALOG BOXES - A DEEP DIVE

Introduction:

Dialog boxes play a crucial role in user interaction, allowing programs to gather additional input beyond simple menus. They typically appear as popup windows containing various child window controls like text boxes, buttons, and radio buttons.

Dialog Box Creation:

Templates: Developers define the layout and appearance of dialog boxes through templates embedded within the program's resource script file. These templates specify the size, position, and type of each control within the dialog box.

Visual Studio: Modern development environments like Visual Studio offer interactive tools for designing dialog boxes. This simplifies the process and generates the corresponding resource script code automatically.

Dialog Box Management:

Windows Responsibility: Upon invocation, Windows 98 takes over the responsibility of creating the dialog box window, its child controls, and a dedicated window procedure to handle messages.

Dialog Box Manager: This internal Windows code manages various aspects of the dialog box, including keyboard and mouse input, and provides the framework for interaction.

Dialog Procedure:

Program-Defined Function: While Windows handles core functionality, developers can implement a custom "dialog box procedure" to perform specific tasks.

Purpose: This procedure typically initializes child controls, processes messages from them (e.g., button clicks), and handles the dialog box's closing.

Focus and Input: Unlike standalone windows, dialog box procedures don't handle WM\_PAINT messages directly or directly process keyboard/mouse input.

Child Window Controls in Dialog Boxes:

Simplified Management: Compared to managing child windows in standalone programs, dialog boxes offer a simpler approach.

Windows Assistance: The built-in dialog box manager takes care of many tasks, including handling focus transition between controls, which was a challenge in Chapter 9.

Building a Simple Dialog Box:

This chapter explores the process of creating and implementing a simple dialog box, showcasing the interplay between the various components involved.

Additional Considerations:

Complexity: While the focus is on a basic example, creating complex dialog boxes with rich features requires more advanced techniques covered later.

Learning Curve: While leveraging child controls within dialog boxes simplifies certain aspects, it introduces new concepts and procedures specific to dialog box interaction.

MODELESS DIALOG BOXES: BEYOND MODALITY

This section delves deeper into the concept of modeless dialog boxes, exploring their characteristics and contrasting them with modal dialog boxes.

Recap: Modal vs. Modeless Dialog Boxes:

Modal: These dialog boxes restrict user interaction to only the dialog box and the program that initiated it. They block access to other windows within the program until closed.

Modeless: These dialog boxes offer greater flexibility by allowing users to switch between the dialog box, the program, and even other applications concurrently.

Benefits of Modeless Dialog Boxes:

Enhanced User Experience: Users can keep the dialog box open for reference while working within the main program, avoiding the need to repeatedly open and close it.

Improved Efficiency: Tasks requiring frequent interaction with both the dialog box and the program are streamlined, minimizing context switching and saving time.

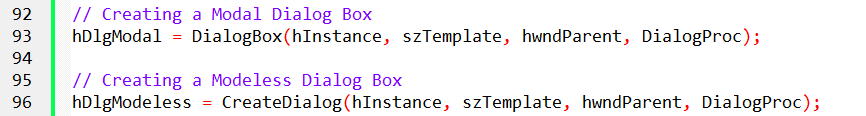
Greater Flexibility: Users can access information displayed in the dialog box while working on other tasks, promoting multitasking and efficient workflow.

Function Comparison: DialogBox vs. CreateDialog

DialogBox: This function is specifically designed for modal dialog boxes. It creates the dialog box, handles user interaction, and only returns after the dialog box is closed.

CreateDialog: This function creates modeless dialog boxes. It returns immediately after creation, handing over the responsibility of managing the dialog box to the program.

Code Comparison:



Remembering the Difference:

The function names provide a clue to their purpose. "DialogBox" emphasizes the box-like nature of modal dialogs, while "CreateDialog" aligns with the creation of regular windows, similar to "CreateWindow".

Additional Considerations:

Modeless dialog boxes require more careful management than modal ones. Developers need to handle closing, responding to user actions, and ensuring the dialog box remains accessible while not interfering with the main window.

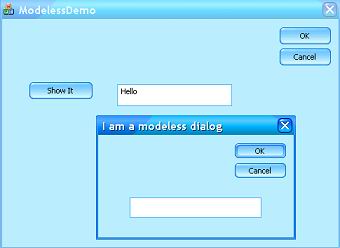
The choice between modal and modeless depends on the specific needs of the application and the intended user interaction.

Deep Dive: Modeless Dialog Boxes and Their Differences

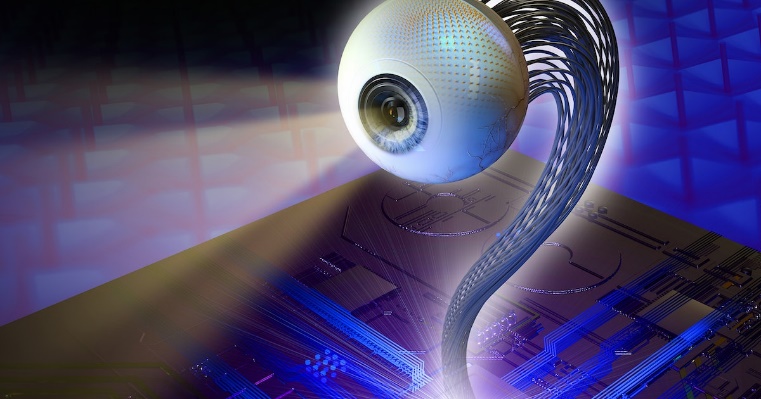
This section delves deeper into the differences between modal and modeless dialog boxes, highlighting key aspects and implementation considerations.

Visual Differences:

Caption Bar and System Menu: Unlike modal dialogs, modeless ones typically include a caption bar for moving the window and a system menu for additional options. This is reflected in the dialog template's STYLE statement, which usually includes WS\_CAPTION and WS\_SYSMENU styles.

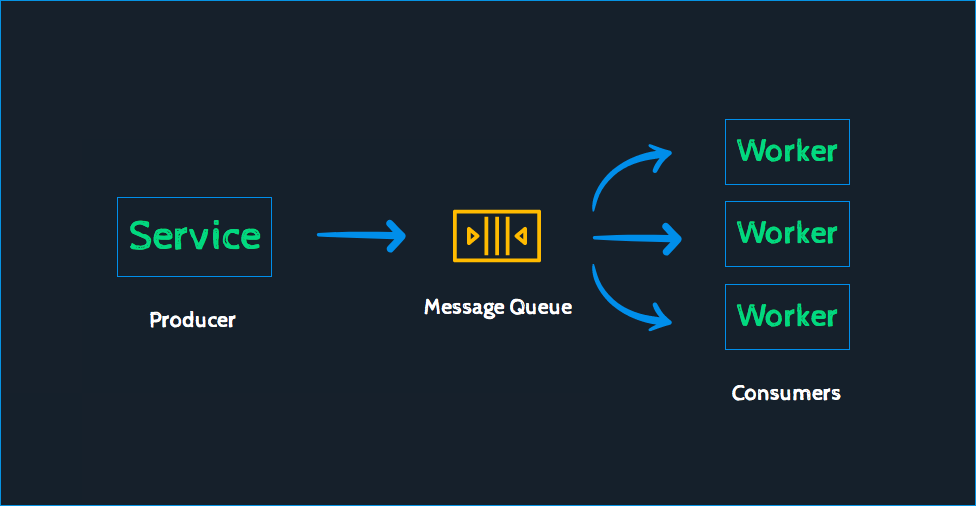


Visibility: Defaulting to hidden, modeless dialog boxes require either the WS\_VISIBLE style in the template or an explicit ShowWindow call with SW\_SHOW to become visible. This differs from modal dialogs which are displayed automatically.

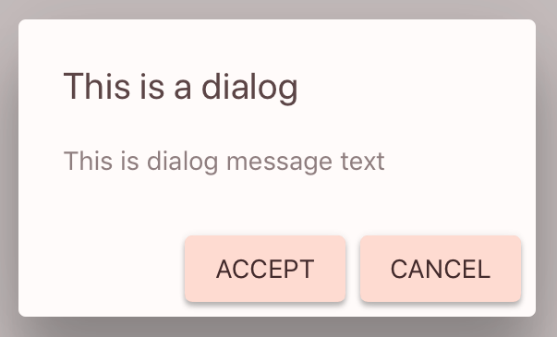


Message Handling:

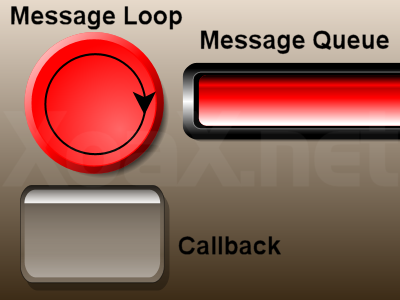
Message Queue: Messages intended for modeless dialog boxes are delivered through the program's message queue, requiring special handling.



IsDialogMessage: This function determines if a message is intended for the modeless dialog box. If so, it sends the message to the appropriate window procedure and returns TRUE. Otherwise, it returns FALSE.



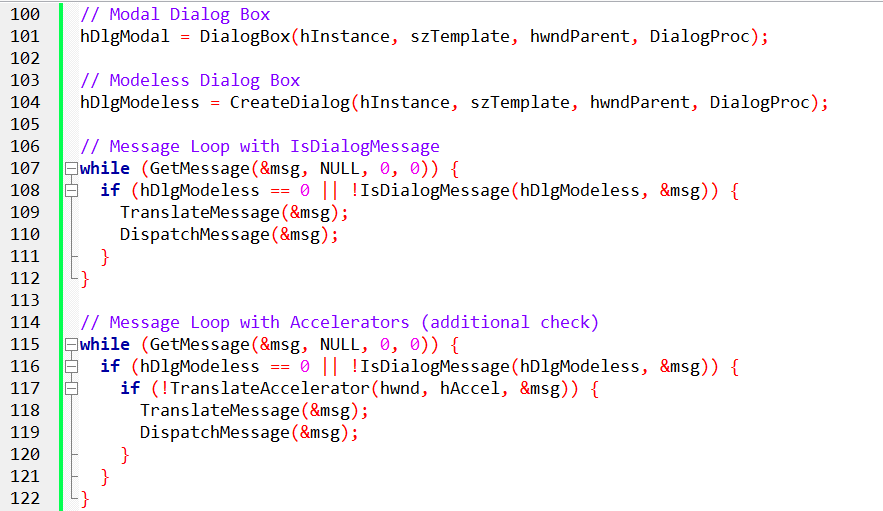
Modified Message Loop: The program's main message loop needs to incorporate IsDialogMessage to filter and dispatch messages accordingly. The basic structure involves checking the dialog box handle (hDlgModeless) and using IsDialogMessage before calling TranslateMessage and DispatchMessage.



Keyboard Accelerators: Programs using keyboard accelerators need to further refine their message loop to ensure proper handling of accelerator messages alongside dialog box messages.



Code Comparison:

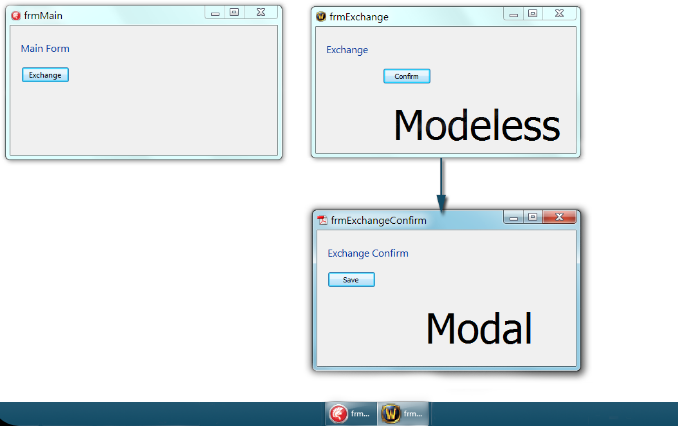


Additional Considerations:

Managing focus: Modeless dialog boxes need careful attention to focus management, ensuring proper focus transitions between the dialog box and other windows.

Memory management: Since the dialog box remains open, proper memory management of the associated window handle is crucial.

User experience: Modeless dialog boxes offer greater flexibility but require careful design to avoid cluttering the desktop and interfering with the main program's functionality.



In this example:

Modal Dialog Box:

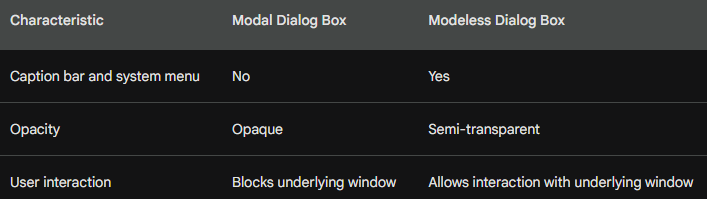
* The dialog box has no caption bar or system menu.
* The dialog box is fully opaque and blocks the underlying window.
* The dialog box must be closed before the user can interact with the underlying window.

Modeless Dialog Box:

* The dialog box has a caption bar and system menu.
* The dialog box is semi-transparent, allowing the underlying window to be partially visible.
* The user can interact with the underlying window while the dialog box is open.

In the image, the "Exchange" dialog box is a modal dialog box, while the "Confirm" dialog box is a modeless dialog box.

Here is a table that summarizes the key differences between modal and modeless dialog boxes:



MASTERING MODELESS DIALOG BOXES: A COMPREHENSIVE GUIDE

This in-depth exploration delves beyond the basics of creating modeless dialog boxes, equipping developers with the knowledge to manage them effectively.

1. The Power of hDlgModeless:

This global variable serves as the central hub for managing the modeless dialog box.

Initialized to 0 by default, it safeguards against invalid handle usage with IsDialogMessage.

Its versatile nature allows for:

* Existence Check: Verifying the dialog box's presence in other program parts.
* Inter-window Communication: Facilitating message exchange between the dialog box and other windows.
* Destruction Control: Identifying the correct handle for proper destruction using DestroyWindow.

2. Ending a Modeless Dialog Box:

Unlike modal dialogs, DestroyWindow replaces EndDialog for closing the dialog box.

Setting hDlgModeless to NULL after DestroyWindow ensures proper memory management.

Users often close the dialog box via the system menu's "Close" option.

The dialog box procedure must capture the WM\_CLOSE message:

* It triggers DestroyWindow to close the dialog box.
* Setting hDlgModeless to NULL completes the destruction process.

3. Push Button Closure:

Similar to handling WM\_CLOSE, push buttons can also initiate closure.

Upon button click, DestroyWindow is called, followed by setting hDlgModeless to NULL.

4. Data Exchange with Parent Window:

Two primary approaches exist for information exchange between the dialog box and the parent window.

* Global Variables: A convenient method for storing data that needs to be "returned" by the dialog box.
* CreateDialogParam: This advanced technique allows passing a structured data pointer for more complex data exchange.

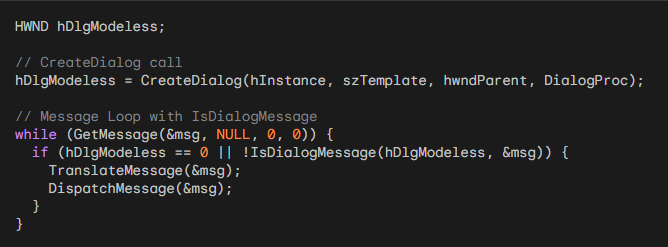
5. Message Loop Orchestration:

IsDialogMessage plays a crucial role in filtering messages intended for the modeless dialog box.

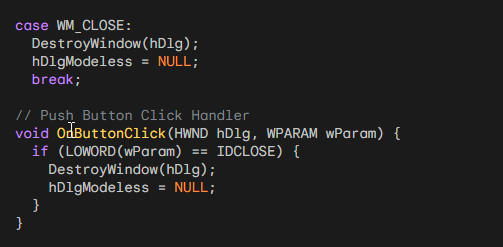
Its integration within the message loop ensures proper message routing.

Handling keyboard accelerators requires combining IsDialogMessage with TranslateAccelerator for seamless interaction.

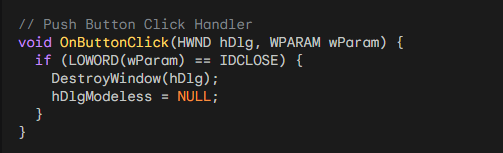
hDlgModeless Global Variable:



Ending a Modeless Dialog Box:

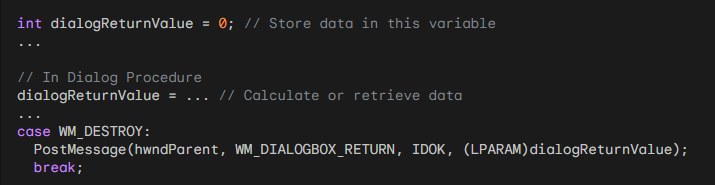


Push Button Closure:

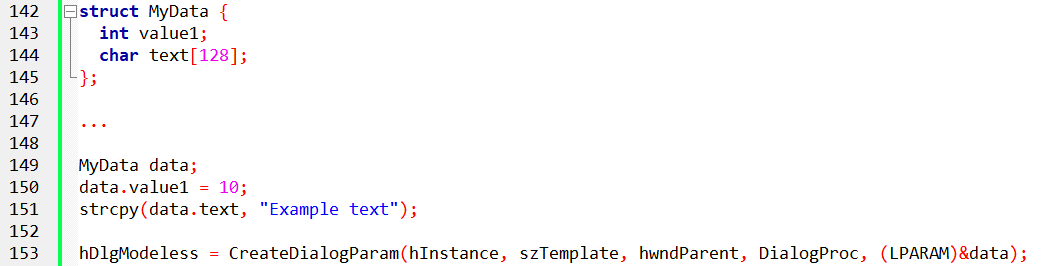


Data Exchange with Parent Window:

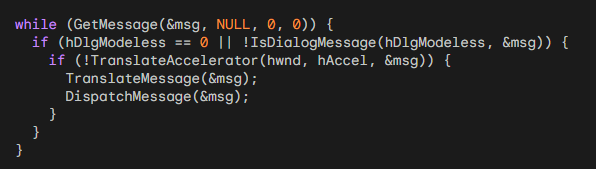
a) Global Variables:



b) CreateDialogParam:



5. Message Loop Orchestration:



Colors2 program in chapter 11 folder….



COLORS2: A Modeless Dialog Box Approach

COLORS2 represents a significant simplification of the COLORS1 program, leveraging the power of modeless dialog boxes. This document dives deep into its functionality and compares it to its predecessor.

COLORS2 utilizes a modeless dialog box to manage the scroll bars and text items previously implemented using child windows in COLORS1.

This approach significantly reduces complexity, particularly within the WndProc function.

The program still allows users to adjust RGB values via scroll bars and displays the corresponding color.

Key Features:

Modeless Dialog Box: The central element of COLORS2 is the "Color Scroll Scrollbars" dialog box. Unlike COLORS1, this dialog box is not modal, allowing users to interact with both the dialog and the main window simultaneously.

Scroll Bar Functionality: Three scroll bars remain, each controlling one RGB color channel (Red, Green, Blue). Users can adjust the values by clicking the arrows, dragging the scroll bar thumb, or entering a value directly in the associated text box.

Dynamic Color Display: As users adjust the scroll bars, the background color of the main window dynamically updates to reflect the chosen RGB combination.

Implementation Compared to COLORS1:

WndProc Simplification: By utilizing a modeless dialog box, the WndProc function in COLORS2 becomes considerably simpler. It now only handles the WM\_DESTROY message for the main window and relies on the dialog box for all scroll bar interactions.

Reduced Complexity: COLORS2 eliminates the need to manage nine child windows individually, reducing the code footprint and complexity.

Message Handling: The program uses IsDialogMessage within the message loop to filter and dispatch messages intended for the dialog box. This ensures efficient message routing.

Code Breakdown:

Main Program: Creates the main window, shows it, and then creates the modeless dialog box using CreateDialog.

Dialog Box: Handles all scroll bar messages (WM\_VSCROLL) and updates the corresponding color values, scroll bar positions, text box values, and background color of the main window.

Resource Script: Defines the layout and appearance of the dialog box, including scroll bars and text boxes.

Benefits of Modeless Dialog Box:

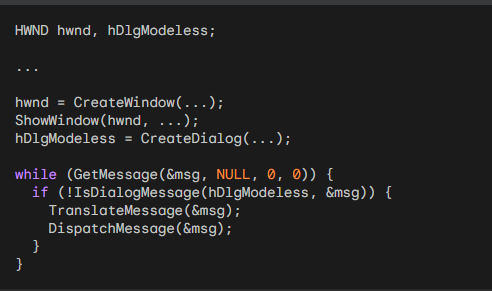
Improved User Experience: Users can adjust color values while still having access to the main window, enhancing workflow and interaction.

Reduced Code Complexity: Managing a single dialog box is simpler and more efficient than handling numerous child windows.

Focus Management: COLORS2 automatically handles focus transitions between the dialog box and other elements, ensuring a smooth user experience.

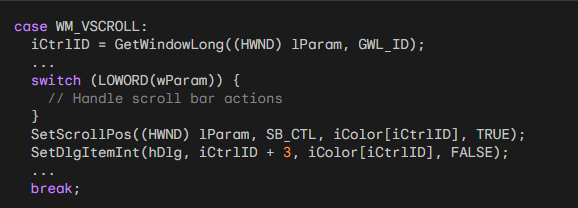
Main Program:

* Creates the main window with the WS\_CLIPCHILDREN style, allowing it to repaint without erasing the dialog box.
* Shows the main window.
* Creates the modeless dialog box using CreateDialog and stores the handle in the global variable hDlgModeless.
* Enters the message loop and filters messages with IsDialogMessage to ensure proper message routing.



Dialog Box:

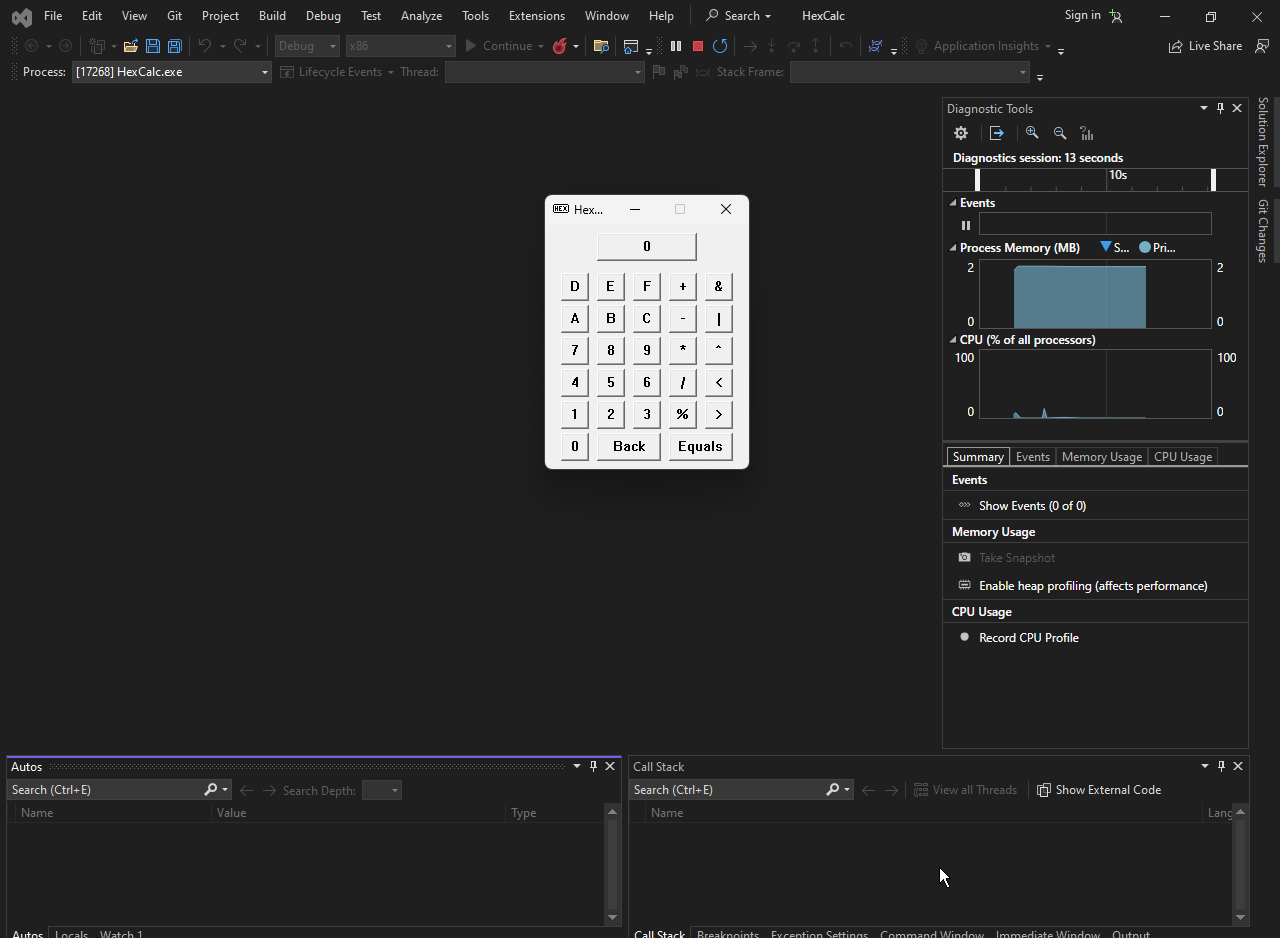
* Handles all scroll bar messages (WM\_VSCROLL) based on their ID numbers (10, 11, and 12).
* Updates the corresponding color values (iColor[i]) based on user interactions.
* Sets the scroll bar positions, text box values, and background color of the main window accordingly.



Resource Script:

* Defines the layout and appearance of the dialog box using controls like scroll bars, static text fields, and labels.

HEXCALC PROGRAM



HEXCALC: A Lazy Programming Marvel

HEXCALC, a program written by Charles Petzold, demonstrates the pinnacle of "lazy programming" by achieving a functional hexadecimal calculator with minimal code.

Instead of employing traditional methods like window creation and message processing, it leverages the power of dialog boxes to simplify its implementation.

Key Features:

* 10-Function Hexadecimal Calculator: Performs basic arithmetic and logical operations on hexadecimal numbers.
* Full Keyboard and Mouse Interface: Supports key presses and button clicks for input and operation selection.
* Minimal Code: Achieved through clever use of dialog boxes and resource files, reducing code complexity and size.
* Dynamic Display: Current number and operation are displayed in real-time.

Technical Highlights:

Dialog Box as the Main Window: Instead of creating a custom window, HEXCALC utilizes a dialog box defined in a resource file. This simplifies window management and reduces code overhead.

Resource Script: Defines the layout and functionality of the calculator interface, including buttons, labels, and text fields.

WndProc Handles Key and Button Events: The WndProc function processes key presses and button clicks, performing calculations and updating the display accordingly.

Message Loop: The program runs in a message loop, receiving and responding to user input and system messages.

Lazy Programming Techniques:

Minimal Window Management: Eliminates the need to create and destroy windows, simplifying the program's logic.

Resource-Driven Interface: Relies on pre-defined resources for the interface, reducing code duplication and maintenance.

Message Loop Efficiency: Processes only relevant messages, minimizing unnecessary processing.

Leveraging Sleep Function: Briefly pauses after button presses to simulate a physical button click, enhancing user experience.

Benefits of This Approach:

Reduced Code Complexity and Size: HEXCALC achieves its functionality in under 150 lines of code, showcasing the power of resource files and message-driven programming.

Faster Development: By relying on pre-built components and efficient coding practices, the program can be developed and implemented quickly.

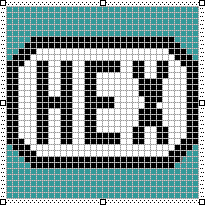
Simplified User Interface: The dialog box interface offers a familiar and user-friendly experience.

Limitations:

Limited Customization: Constrained by the capabilities of dialog boxes, customizability of the interface is limited.

Resource Dependence: Relies heavily on resource files for functionality, requiring additional maintenance alongside the code.

Potentially Less Efficient: May not be as efficient or performant as programs built with more traditional methods.



HEXCALC is a unique program that blurs the lines between traditional windows and modeless dialog boxes. It functions as a standard infix notation calculator for unsigned 32-bit integers, offering basic arithmetic, logical operations, and bit shifts. Key Features:

* Hybrid Window/Dialog Box: HEXCALC utilizes a dialog box template (HEXCALC.DLG) for its interface, yet its functionality is handled by a custom window procedure (WndProc) similar to traditional windows.
* Infix Notation: Users can enter operations and operands in familiar infix notation, similar to a standard calculator.
* 32-Bit Integer Support: Performs calculations on unsigned 32-bit integers, covering a wide range of values.
* Basic Operations: Supports addition, subtraction, multiplication, division, and remainder calculations.
* Bitwise Operations: Offers bitwise AND, OR, and exclusive OR operations for manipulating individual bits.
* Bit Shifts: Allows for left and right shifting of bits within the 32-bit integer.

User Interface and Interaction:

* Mouse and Keyboard Input: Users can interact with the calculator using both mouse clicks on buttons and keyboard input for numbers and operations.
* Display Box: A designated "display" box shows the current entry and the calculated result.
* Clear and Backspace: The "Back" button and keyboard keys like Backspace and Left Arrow allow for correcting mistakes in the entry.
* Result Display: Clicking the "Equals" button or pressing Enter key shows the final result of the calculation.

Behind the Scenes:

* Dialog Box Template: The HEXCALC.DLG file defines the layout and functionality of the calculator interface, including buttons, labels, and text fields.
* Custom Window Procedure: The WndProc function handles all messages sent to the window, including key presses, button clicks, and calculations.
* Message Loop: The program runs in a message loop, receiving and responding to user input and system messages.

Inner working of HexCalc:

HEXCALC is a unique program that combines the characteristics of both traditional windows and modeless dialog boxes. This section delves deeper into its inner workings, highlighting key aspects of its design and implementation.

Dialog Box Template and Window Procedure:

Crucial Difference: While HEXCALC utilizes a dialog box template (HEXCALC.DLG) for its interface, its functionality is managed by a custom window procedure (WndProc) similar to a traditional window.

Overlapping Dialog: The presence of the CLASS statement in the HEXCALC.DLG template distinguishes it from standard dialog boxes. This statement instructs Windows to send messages to the HexCalc window class instead of using its internal window procedure.

Window Class Registration: Similar to a normal window, HEXCALC registers the HexCalc window class in its WinMain function. However, it sets the cbWndExtra field of the WNDCLASS structure to DLGWINDOWEXTRA, which is crucial for handling messages through a custom window procedure.

Dialog Box Creation and Message Handling:

CreateDialog Call: WinMain calls CreateDialog to create the window. This function effectively translates into a CreateWindow call, creating the main window and its 29 child button controls.

Child Window Management: Windows automatically handles creating these child windows and sending WM\_COMMAND messages to the WndProc. This simplifies the process of managing a window with multiple child controls.

Minimal Code Size: HEXCALC leverages the ASCII codes of button text as their IDs, eliminating the need for a separate header file with control identifiers. This allows WndProc to handle both WM\_COMMAND and WM\_CHAR messages efficiently.

Keyboard Interaction and Message Processing:

Left Arrow and Backspace: WndProc intercepts WM\_KEYDOWN messages and converts the Left Arrow key to a Backspace key for consistent behavior.

Keyboard Character Conversion: During WM\_CHAR message processing, WndProc converts the character code to uppercase and translates the Enter key to the Equals key for intuitive user experience.

Validating Keyboard Input: GetDlgItem ensures the received character corresponds to a button ID defined in the dialog box template. This avoids processing invalid keyboard input.

Button Flashing and User Feedback: WndProc utilizes BM\_SETSTATE messages to visually indicate button clicks by flashing them for 100 milliseconds, enhancing user feedback.

Main Window Focus:

Maintaining Focus: Notably, WndProc explicitly sets the input focus back to the main window after handling WM\_COMMAND messages. This ensures keyboard input is directed to the main window, preventing accidental focus shifts to buttons.

Conclusion:

HEXCALC serves as a prime example of "lazy programming" done right. By cleverly utilizing dialog boxes and resource files, it achieves a functional and user-friendly application with minimal code, demonstrating the power of efficient coding practices.

While this approach may not be suitable for all applications, it offers a valuable alternative for quick and simple development of small, user-driven programs.

The HEXCALC.DLG file cannot be directly edited through the Visual Studio Dialog Editor. It requires manual editing to include specific functionalities.

The resource script needs to be modified to include the HEXCALC.DLG file using the #include directive.

Understanding the message loop and WndProc function is crucial for modifying or extending the program's functionality.

THE COMMON DIALOG BOXES: A REVOLUTION IN STANDARDIZATION

The standardized user interface was a key objective of Windows from its inception. While achieving uniformity across various software for common menu items like "Alt-File-Open" was rapid, the actual file open dialog boxes remained diverse.

Common Dialog Box Library: A Standardized Solution

Windows 3.1 introduced the "common dialog box library," a revolutionary solution to the problem of inconsistent dialog boxes. This library provides functions that invoke standard dialog boxes for various tasks, including:

* Opening and saving files
* Searching and replacing text
* Choosing colors
* Selecting fonts (demonstrated in this chapter)
* Printing (demonstrated in Chapter 13)

Functionalities and Usage:

* Structure Initialization: Before invoking the dialog box, you initialize the fields of a specific structure relevant to the desired functionality.
* Function Call: Pass a pointer to this structure to a function in the common dialog box library.
* Dialog Box Display: The function creates and displays the standard dialog box for the specified task.
* User Interaction: The user interacts with the dialog box and makes their selections.
* Function Return: When the dialog box closes, the function returns control to your program.
* Information Retrieval: You retrieve information about the user's choices from the structure you passed to the function.

Header File and Reference Documentation:

* Include the COMMDLG.H header file in your C source code to use the common dialog box library.
* Refer to the documentation provided at /Platform SDK/User Interface Services/User Input/Common Dialog Box Library for detailed information on each function and its associated structure.

POPPAD Revisited: Implementing Common Dialog Boxes

Previously in Chapter 10, we added a menu to POPPAD but left several standard menu options unimplemented. Now, we'll enhance POPPAD3 to incorporate functionality for:

* Opening files: Utilize the common dialog box library to open existing text files.
* Saving edited files: Allow users to save their edits back to disk using the common dialog box.
* Selecting fonts: Provide the ability to change the font used for displaying text within POPPAD.
* Searching and replacing text: Implement functionality to search and replace text within the document.

POPPAD3 Program Resources: