CHAPTER 19: MULTIPLE DOCUMENT INTERFACE

MDI (Multiple-Document Interface):

It's a design approach for applications that manage multiple documents in Microsoft Windows.

It allows users to work with several documents simultaneously within a single application window.

Think of it like having multiple tabs open in your web browser, but for different documents within the same program.

Key concepts:

* Parent window: The main application window that holds all the document windows.
* Child windows: The individual document windows that reside within the parent window.
* Client area: The part of the parent window where the child windows are displayed.

How MDI works:

Opening documents: Each document opens in its own child window within the parent window.

Arranging windows: Users can arrange child windows in various ways:

* Tile them horizontally or vertically to view multiple documents side-by-side.
* Cascade them to overlap like a stack of cards.
* Arrange them manually by dragging and resizing.

Switching between windows: Users can switch between documents by clicking on the desired child window, or using keyboard shortcuts.

Managing windows: The application provides features to manage child windows, such as:

* Arranging them (as mentioned above).
* Minimizing, maximizing, or closing them.

Benefits of MDI:

Efficient multitasking: Enables working on multiple documents simultaneously without cluttering the desktop with separate application windows.

Easy comparison: Allows side-by-side comparison of documents for easier referencing and editing.

Organization: Helps keep related documents grouped together within a single application.

Examples of MDI applications:

* Older versions of Microsoft Office applications (Word, Excel, PowerPoint)
* Adobe Photoshop
* Many text editors and code editors

Modern trends:

While MDI was common in older Windows applications, newer applications often use alternative approaches like:

* Tabbed interfaces (similar to web browsers)
* Single-document interfaces (SDI) where each document opens in its own separate window

These approaches offer different user experiences and trade-offs in terms of window management and organization.

MDI in Historical Context:

Early Complexity: While MDI was introduced in Windows 2.0, its implementation was challenging for developers due to intricate programming requirements.

Greater Support: Windows 3.0 and subsequent versions significantly streamlined MDI development by providing built-in support and enhancements.

Key Elements of MDI Applications:

Main Application Window:

Similar to standard application windows, featuring a title bar, menu, sizing border, system menu icon, minimize/maximize/close buttons.

Unique client area, often called the "workspace," designed specifically to house child windows instead of direct program output.

Child Windows (Document Windows):

Resemble small application windows within the main window's workspace.

Possess title bars, sizing borders, system menu icons, minimize/maximize/close buttons, and potentially scroll bars.

Notably lack their own menus, relying on the main application window's menu for actions.

Active Document:

Only one child window can be active at a time, indicated by a highlighted title bar.

Active window appears in front of other child windows, ensuring focus.

Document Containment:

Child windows are confined to the workspace area within the main application window, preventing them from extending beyond its boundaries.

Essential Programming Considerations:

* Window Management: MDI support involves specific functions and data structures for managing the relationship between the main application window and its child windows.
* Message Handling: MDI applications must process window messages differently to coordinate interactions between the main window and child windows.
* User Experience: Programmers must carefully design the MDI interface to ensure intuitive navigation, window management, and document interaction for users.

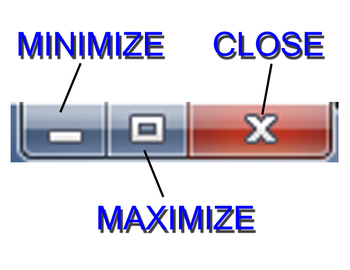
BEYOND INITIAL IMPRESSIONS:

While Multiple-Document Interface (MDI) might seem straightforward at first glance, it quickly reveals a nuanced set of complexities that demand careful attention from developers. Let's delve deeper into the key elements and considerations involved in crafting effective MDI applications.

Unveiling the Nuances:

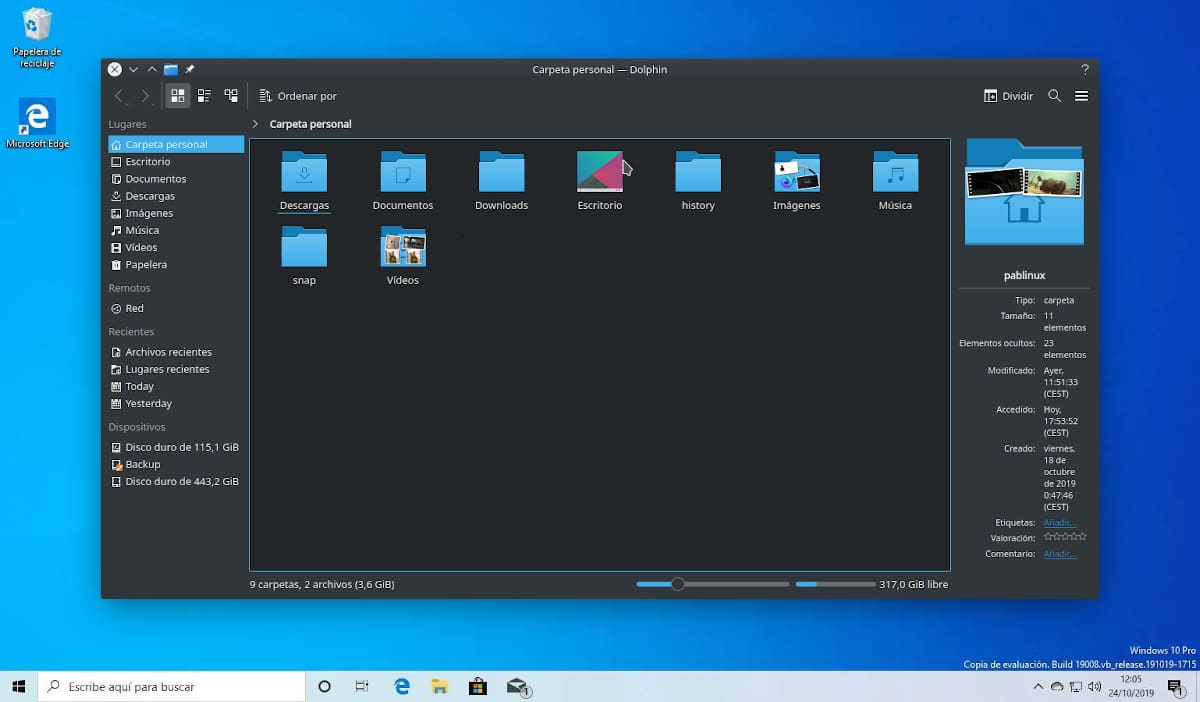
Minimized Windows:

* These windows gracefully tuck themselves away as compact title bars with icons, residing at the bottom of the workspace.
* To aid visual clarity, they typically adopt distinct icons from the main application window, ensuring easy differentiation.



Maximized Windows:

* In this state, they seamlessly blend their title bars with the main window's, effectively appending document filenames for a cohesive display.
* System menu icons and close buttons relocate to the main window's menu bar, maintaining a unified interface while preserving access to essential controls.



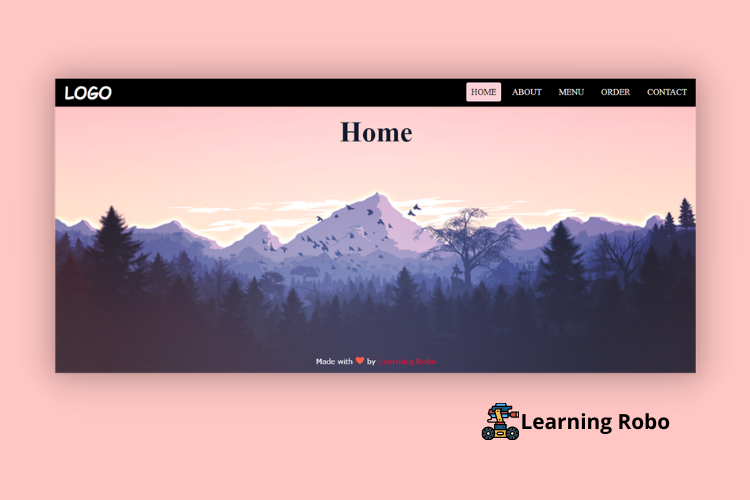
Keyboard Shortcuts for Efficiency:

* Ctrl+F4 offers a swift way to close document windows, while Alt+F4 retains its traditional role of closing the main application window.
* Ctrl+F6 enables effortless switching between child windows, promoting fluid navigation.
* Alt+Spacebar remains dedicated to invoking the main window's system menu, and Alt+- (minus) unlocks the active child window's system menu for granular control.



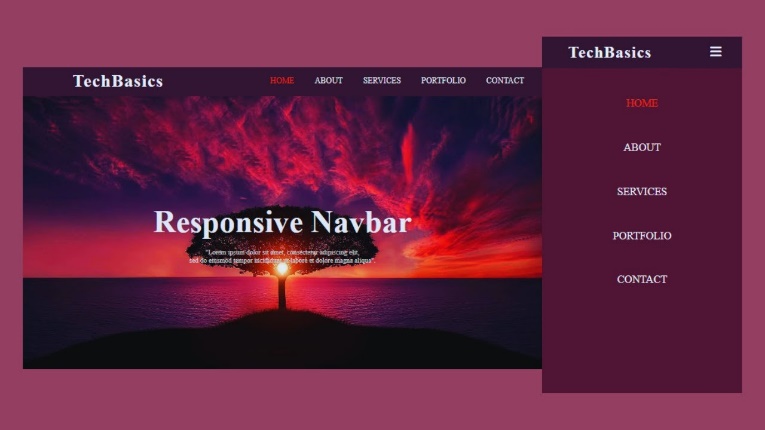
Menu Navigation: A Unified Journey:

* Cursor keys gracefully guide users through MDI menus, initiating their journey at the application system menu, then gracefully transitioning to the active document system menu, before finally arriving at the first item on the main menu bar. This intuitive flow promotes a cohesive user experience.



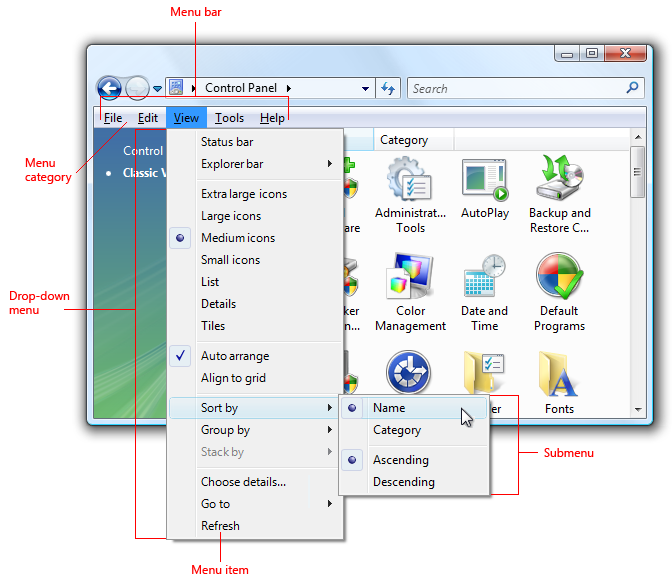
Context-Aware Menus:

* Intelligent applications dynamically adapt their menus based on the active document's type, ensuring that only relevant actions are presented.
* This responsiveness extends to the absence of document windows, where menus gracefully streamline to showcase options for opening or creating new documents, minimizing clutter and guiding users towards primary tasks.



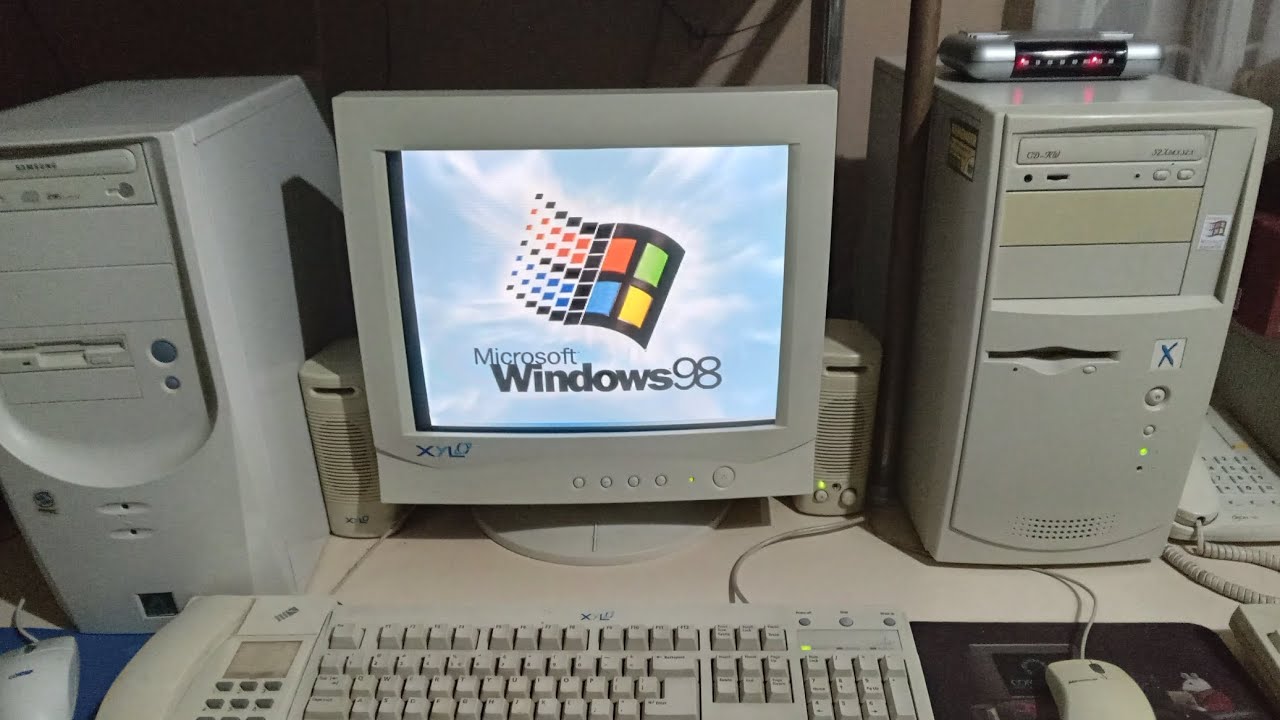
The Window Menu: A Command Center for Document Management:

* Nestled strategically within the top-level menu bar, typically preceding the Help menu, this dedicated menu empowers users to effortlessly arrange and access document windows.
* It offers convenient options for cascading windows in an overlapping fashion or tiling them for full visibility, catering to different organizational preferences.
* A comprehensive list of all open document windows resides within this menu, enabling users to swiftly select and activate desired documents with a single click.



Windows 98: A Helping Hand:

* Windows 98 extended a valuable hand to developers by incorporating built-in support for these MDI features, significantly reducing the programming effort required for their implementation.
* While this support does introduce some overhead, it's a small price to pay compared to the manual implementation of such a comprehensive set of functionalities.



MDI: A FAMILY OF WINDOWS

To create a well-structured Multiple-Document Interface (MDI) application, it's crucial to understand the relationships between its various window types:

1. Frame Window:

The majestic patriarch of the MDI family, this window serves as the main application window.

It possesses the WS\_OVERLAPPEDWINDOW style, granting it the familiar features of a standard Windows window, including a title bar, menu, sizing border, and system buttons.

2. Client Window:

This loyal servant, a child of the frame window, is crafted using the predefined MDICLIENT window class.

It's given life through a call to CreateWindow with the WS\_CHILD style, and it dutifully covers the frame window's client area like a protective cloak.

The CLIENTCREATESTRUCT structure, passed as an argument to CreateWindow, provides additional guidance for its creation.

Its color, derived from the system color COLOR\_APPWORKSPACE, often blends seamlessly with the frame window, creating a unified workspace for child windows.

3. Child Windows (Document Windows):

These diligent offspring, representing individual documents, reside within the client window's workspace.

To bring them into existence, an MDICREATESTRUCT structure is meticulously prepared and sent to the client window via a WM\_MDICREATE message.

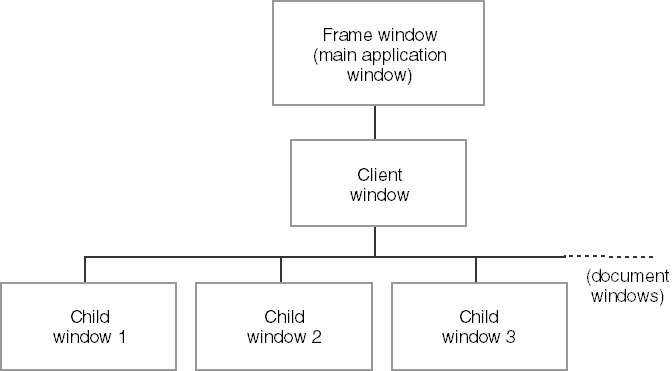
They lack their own menus, deferring to the frame window's menu for actions.

The MDI Hierarchy:

* A clear chain of command exists within the MDI realm:
* The frame window reigns supreme as the parent of the client window.
* The client window, in turn, serves as the parent of all child windows.
* This well-defined hierarchy ensures order and cooperation among the windows, enabling a cohesive user experience.

Visualizing the Family Tree:

Figure 19-1 elegantly captures this hierarchy, depicting the frame window as the root, the client window as its branch, and the child windows as leaves sprouting from that branch.



The image depicts the hierarchy of an MDI application. Here's a breakdown of the image and how it relates to the text I previously provided:

* Frame window: This is the main application window, represented at the top of the image as "Frame window (main application window)". It has the familiar elements of a standard Windows window, such as a title bar, menu bar, and sizing border.
* Client window: This window sits beneath the frame window and acts as a container for the child windows. The text describes it as the "client window (MDICLIENT)" and mentions that it covers the client area of the frame window.
* Child windows: These are the individual document windows, shown in the image as "Child window 1", "Child window 2", and "Child window 3". They are all children of the client window and reside within its workspace.

Key Points:

* The client window, while visually subtle, plays a pivotal role in orchestrating MDI functionality.
* Child windows, lacking their own menus, rely on the frame window's menu for user interactions.
* The WM\_MDICREATE message is the key to birthing new child windows into the MDI world.

ESSENTIAL ELEMENTS FOR MDI DEVELOPMENT:

Window Classes and Procedures:

Frame Window: Requires its own window class and window procedure to handle events and interactions.

Child Windows: Each type of child window also demands a distinct window class and procedure for unique behaviors.

Client Window: Relies on the pre-registered MDICLIENT class, eliminating the need for a custom window procedure.

MDI Support Components:

Window Class:

* MDICLIENT: The pre-registered class for the client window, responsible for core MDI functionality.

Data Structures:

* CLIENTCREATESTRUCT: Used for client window creation, providing initial configuration.
* MDICREATESTRUCT: Employed for child window creation, defining their properties and behaviors.

Functions:

* DefFrameProc: Replaces DefWindowProc for frame window procedures, handling MDI-specific messages.
* DefMDIChildProc: Substitutes DefWindowProc for child window procedures, managing MDI child window actions.
* TranslateMDISysAccel: Interprets system accelerator keys within the MDI context (similar to TranslateAccelerator).
* CreateMDIWindow: Optional for multi-threaded programs, enabling child window creation in separate threads.

Messages:

* Twelve WM\_MDI messages facilitate communication between frame, client, and child windows for tasks like creation, activation, arrangement, and information retrieval.

Key Points:

* Frame Windows Initiate MDI Actions: Frame windows typically initiate MDI operations by sending WM\_MDI messages to the client window.
* Client Window Orchestrates Child Windows: The client window handles WM\_MDI messages and coordinates actions among child windows.
* WM\_MDIACTIVATE Exception: This message is both sent by the frame window to activate a child window and by the client window to inform affected child windows about activation changes.

Building an MDI Application:

1. Register custom window classes for the frame window and any child window types.
2. Create the frame window using CreateWindow.
3. Within the frame window procedure, create the client window using CreateWindow with the MDICLIENT class.
4. Use WM\_MDICREATE messages to create child windows as needed.
5. Employ DefFrameProc and DefMDIChildProc for appropriate message handling.
6. Utilize TranslateMDISysAccel for accelerator key translation.
7. Optionally use CreateMDIWindow for child window creation in separate threads.

Remember:

* MDI support simplifies complex MDI application development.
* Understanding these components is crucial for effective MDI implementation.

THE MDIDEMO PROGRAM

Frame Window Procedure (FrameWndProc):

Client Window Creation:

* Employs "MDICLIENT" class for client window, leveraging built-in MDI functionality.
* Utilizes CLIENTCREATESTRUCT to tailor window menu and child window IDs, ensuring a cohesive user experience.

Child Window Handling:

* Manages creation and destruction of child windows via WM\_MDICREATE and WM\_MDIDESTROY messages, effectively coordinating multiple document interfaces.
* Forwards unhandled commands to active child windows using WM\_COMMAND, promoting flexibility and customization within child windows.

Termination Handling:

* Gracefully attempts to close all child windows during session termination or frame window closure, ensuring data integrity and user experience.

Message Routing:

* Employs DefFrameProc for specialized MDI message handling, streamlining MDI-specific behaviors within the frame window.

Client Window Messages:

Child Window Management:

* WM\_MDICREATE and WM\_MDIDESTROY handle child window lifecycle, ensuring efficient resource allocation and deallocation.
* WM\_MDIACTIVATE controls child window focus, enabling seamless user interactions within the MDI environment.
* WM\_MDIGETACTIVE retrieves the active child window, facilitating targeted operations and communication.
* WM\_MDITILE, WM\_MDICASCADE, and WM\_MDIICONARRANGE provide visual organization of child windows, enhancing user experience and workspace management.
* Additional messages like WM\_MDINEXT, WM\_MDIMAXIMIZE, and WM\_MDITILE manage child window positions and states, offering a comprehensive toolkit for window manipulation.

Child Window Procedures:

*Window-Specific Data:*

* Utilize WM\_CREATE to allocate and store window-specific data using SetWindowLong, enabling tailored functionality and persistence of settings.
* Maintain handles to client and frame windows for efficient communication and coordination within the MDI hierarchy.

*Message Handling:*

* Process child window-specific commands in WM\_COMMAND, allowing for unique behaviors and interactions within each child window.
* Perform child window painting in WM\_PAINT, ensuring proper visual representation and responsiveness.
* Update menus based on activation status in WM\_MDIACTIVATE, maintaining context-specific menu options for user convenience.
* Prompt for confirmation before closing in WM\_QUERYENDSESSION and WM\_CLOSE, preventing accidental data loss and promoting user control.
* Free resources and data in WM\_DESTROY, ensuring proper memory management and preventing leaks.

*Message Routing:*

* Employ DefMDIChildProc for specialized MDI message handling within child windows, ensuring adherence to MDI conventions and behaviors.

Memory Management:

* Efficiently manages memory allocation and deallocation during window creation and destruction using HeapAlloc and HeapFree.
* Ensures proper handling of window-specific data associated with each instance, preventing memory leaks and promoting resource efficiency.

Timer Usage in RectWndProc:

* Utilizes a timer (SetTimer) in the RectWndProc to periodically generate random rectangles.
* Adds a dynamic element to the Rect child window, enhancing visual appeal and user engagement.

Menu Initialization:

* Initializes menus and submenus for various window types during program startup.
* Ensures correct association of menus with the MDI client window, providing a seamless and intuitive user interface.

Window Registration:

* Registers window classes for the frame window, Hello child window, and Rect child window.
* Facilitates proper handling of different window types within the application, contributing to a well-organized and modular structure.

Accelerator Key Translation:

* Incorporates TranslateMDISysAccel to streamline system accelerator key translation within MDI windows.
* Enhances keyboard navigation and user experience by providing consistent and expected behavior for system-defined key combinations.

Structured Window Creation Parameters:

* Utilizes CLIENTCREATESTRUCT and MDICREATESTRUCT to define client window properties and child window creation parameters in a structured manner.
* Promotes code clarity and maintainability by organizing window-related information in a systematic format.

DefFrameProc and DefMDIChildProc:

* Employs DefFrameProc for specialized MDI message handling within the frame window.
* Utilizes DefMDIChildProc for specialized MDI message handling within child windows.
* Ensures adherence to MDI conventions and behaviors, simplifying the implementation of MDI-compliant features.

Additional Insights:

* TranslateMDISysAccel: Streamlines system accelerator key translation within MDI windows, enhancing keyboard navigation and user experience.
* CLIENTCREATESTRUCT and MDICREATESTRUCT: Offer structured approaches to defining client window properties and child window creation parameters, promoting clarity and maintainability.
* DefFrameProc and DefMDIChildProc: Expose specialized message handling functions for MDI frame and child windows, simplifying MDI-compliant behavior implementation.