

# 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.

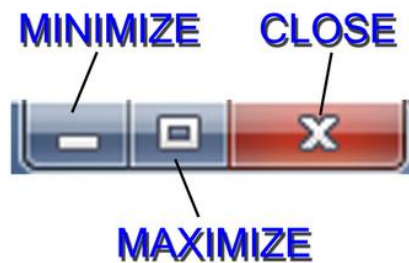
# BEYOND INITIAL IMPRESSIONS:

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:

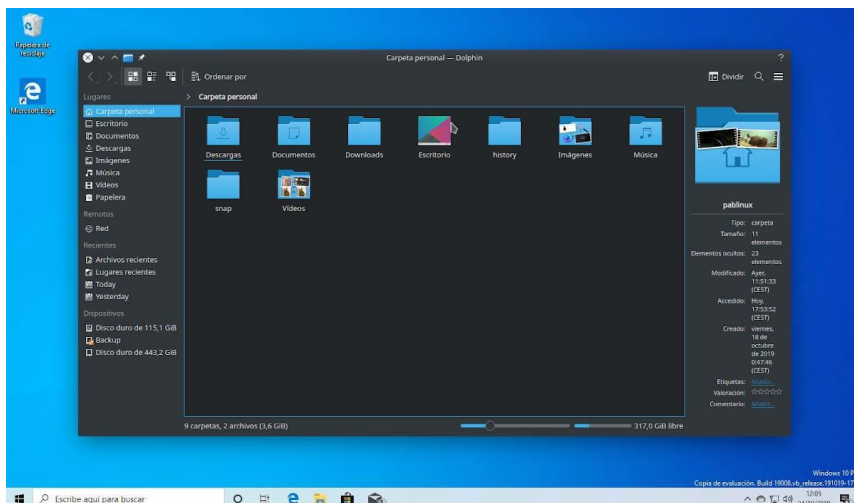
### Minimized Windows:

- 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.



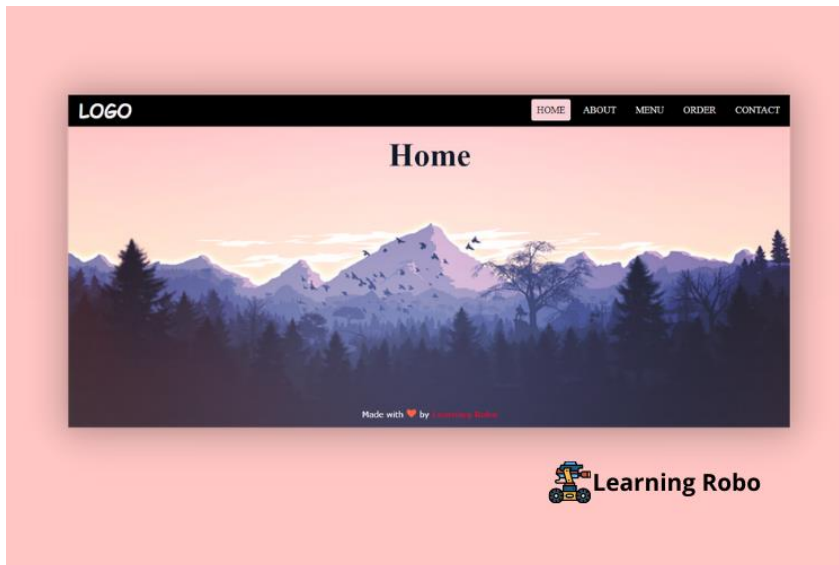
## Keyboard Shortcuts for Efficiency:

- 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.



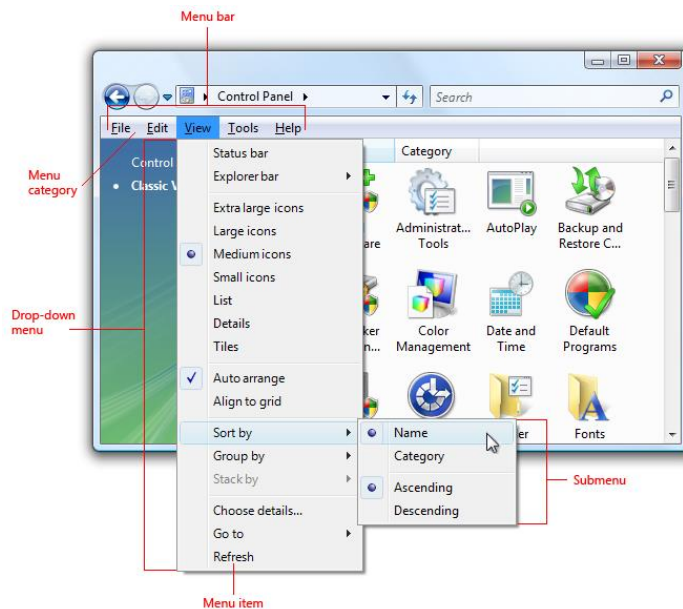
## Context-Aware Menus:

- 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.

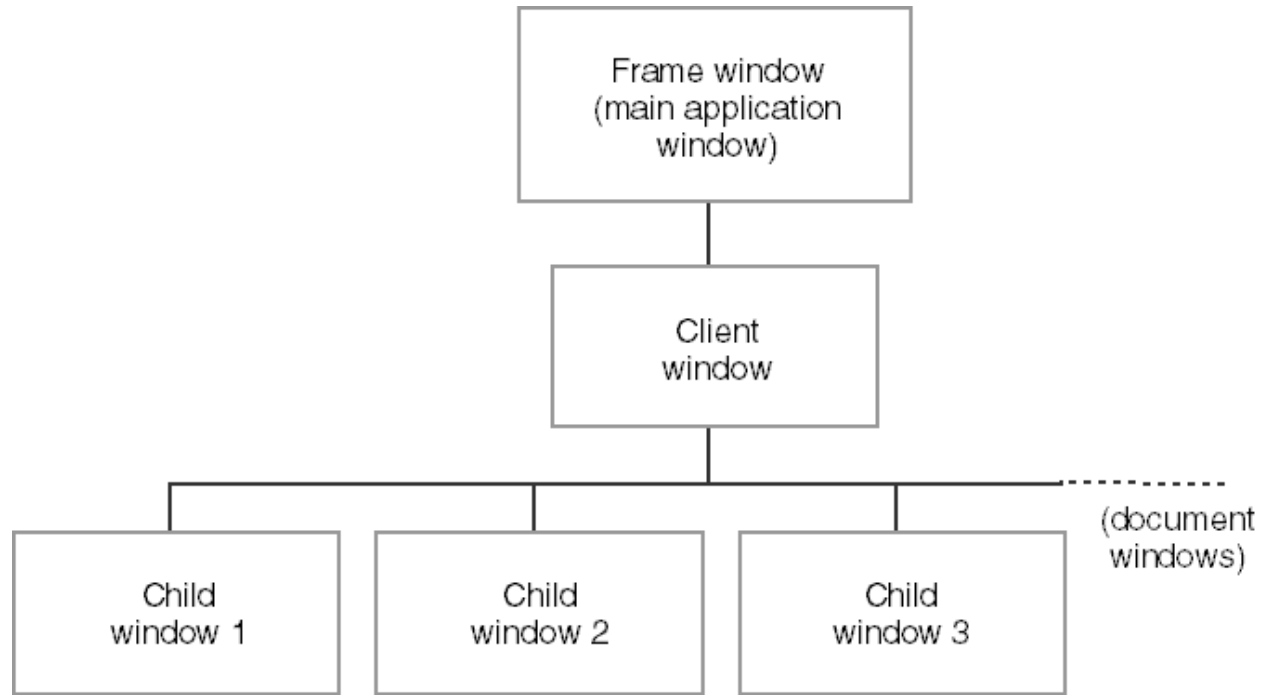
## The MDI Hierarchy:

- 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:

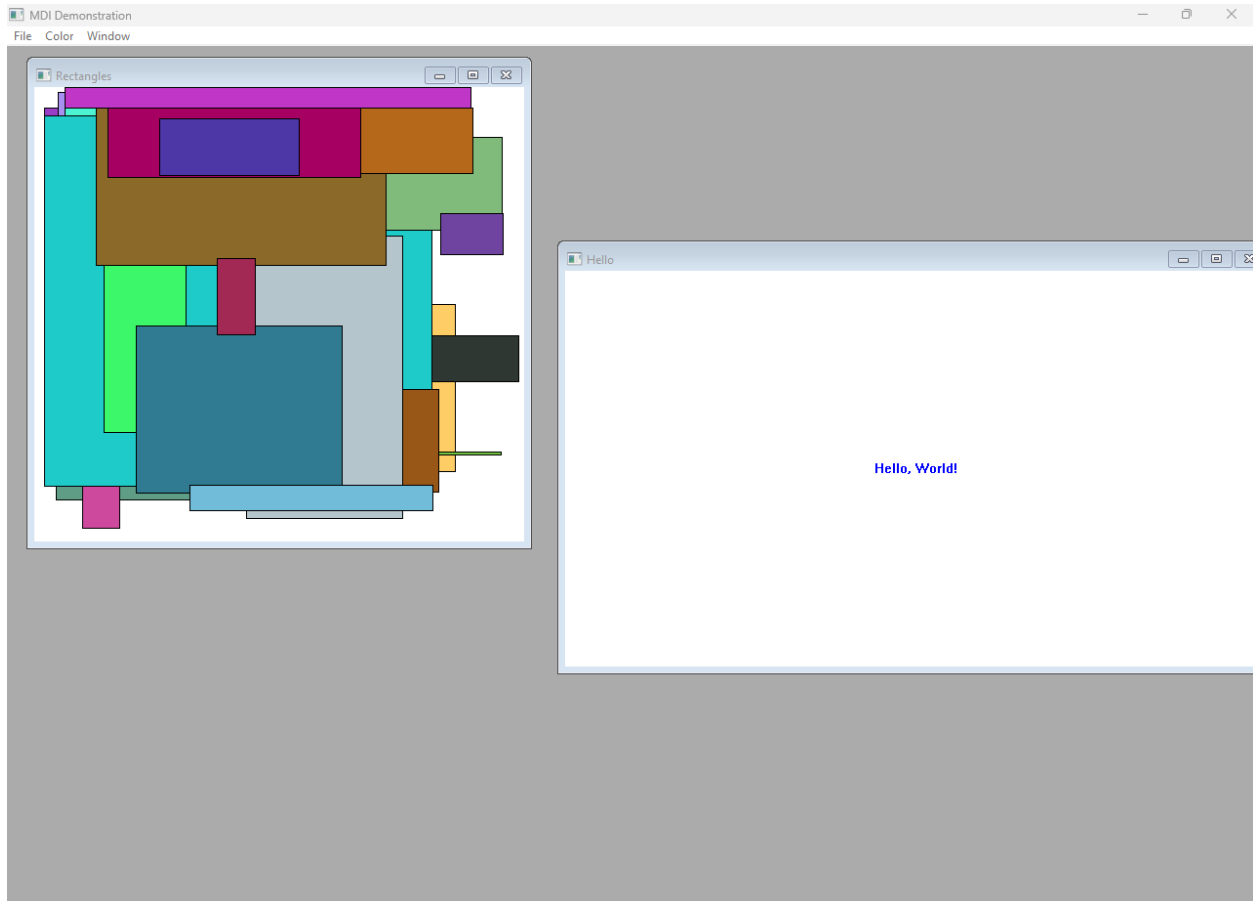
- ❖ Incorporates [TranslateMDISysAccel](#) to streamline system accelerator key translation within MDI windows.
- ❖ [Enhances keyboard navigation](#) and user experience by providing consistent and expected behavior for system-defined key combinations.

## 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.

The program in action...



MDI-Demo  
Program.mp4



# KEY MENU COMPONENTS:

## Three Menu Templates:

**MdiMenuInit:** Initial menu for empty workspace, offering options to create new documents or exit.

**MdiMenuHello:** Menu for "Hello, World!" document windows, featuring color-changing options and window-arranging controls.

**MdiMenuRect:** Menu for random rectangle document windows, similar to MdiMenuHello but without color options.

## Menu Identifiers:

- **RESOURCE.H** header file defines all menu IDs.
- **INIT\_MENU\_POS, HELLO\_MENU\_POS, RECT\_MENU\_POS:** Constants indicating the position of the Window submenu within each menu template, used to inform the client window where to display the document list.
- **IDM\_FIRSTCHILD:** Identifier for the first document window in the Window submenu, ensuring unique IDs for menu items associated with child windows.

## Dynamic Menu Behaviors:

### Document List in Window Submenu:

- The Window submenu dynamically lists open document windows, providing easy access and management.
- The program correctly positions this list within the appropriate menu template, even for MdiMenuInit (which lacks a visible Window submenu).

### Menu Changes Based on Active Window:

- The displayed menu switches based on the active child window, ensuring context-sensitive options and streamlined user experience.

## Additional Considerations for MDI Menus:

### Message Routing:

The frame window often handles menu-related messages before forwarding unhandled commands to active child windows, enabling centralized control and coordination.

### Accelerator Keys:

Consider using `TranslateMDISysAccel` to enhance keyboard navigation within MDI windows, optimizing user interactions.

### Message Handling:

Child window procedures typically handle their specific menu commands within `WM_COMMAND` message processing, ensuring tailored functionality.

### Menu Updates:

Child windows often update menus during `WM_MDIACTIVATE` to reflect their active/inactive status and provide context-appropriate options, maintaining a consistent user interface.

## WINDOW CLASS ORCHESTRATION:

The program **carefully creates window classes** for different types of windows: the main frame window, the "Hello, World!" document windows, and the random rectangle document windows. Each window class has its own function to handle messages and events.

The **background color of the frame window** is set to `COLOR_APPWORKSPACE` to make it visually consistent with the client window.

All **three window classes allocate additional space using the `cbWndExtra` field** in the `WNDCLASS` structure. This extra space is important for storing data specific to each window, allowing for customization and flexibility in the document windows. Menu Management and User Interaction:

The program **eagerly loads three distinct menus**: `MdiMenuInit`, `MdiMenuHello`, and `MdiMenuRect`. Each menu is designed to cater to specific scenarios within the application's workflow.

The **Window submenus**, which dynamically list open document windows, are meticulously identified and their handles are stored for efficient access.

An **accelerator table** is loaded to enhance keyboard navigation and streamline user interactions within the application.

## Frame Window Construction:

The **WinMain function** diligently creates the frame window using the `CreateWindow` function. This establishes the foundational structure for the MDI application.

During the **WM\_CREATE message processing**, the frame window gracefully orchestrates the creation of the client window. This client window, belonging to the preregistered `MDIClient` class, serves as the cornerstone for MDI functionality.

The **client window procedure, acting as a crucial intermediary** between the frame window and the document windows, is configured with a `CLIENTCREATESTRUCT` structure. This structure thoughtfully specifies:

The **handle of the Window submenu**, initially set to `hMenuInitWindow`.

The **menu ID** to be associated with the first document window, designated as `IDM_FIRSTCHILD`.

## Message Loop Mechanics:

The application displays the newly constructed frame window for user interaction.

A tailored message loop, specifically designed for MDI applications, commences to process events:

- **Incoming messages are first scrutinized** by `TranslateMDISysAccel` to seamlessly translate MDI-specific accelerator keys, such as `Ctrl-F6` for window switching.
- **If menu accelerators are present**, they are gracefully handled by `TranslateAccelerator`.
- Should either **translation function** indicate success, the message is deemed fully processed, and the standard `TranslateMessage` and `DispatchMessage` calls are strategically bypassed to ensure efficiency.
- The **hwndClient** and **hwndFrame window handles** are thoughtfully passed to the translation functions, empowering them with the necessary context for accurate message interpretation.
- The **hwndClient handle**, representing the client window, is meticulously obtained using the `GetWindow` function with the `GW_CHILD` argument, ensuring a clear reference to the MDI client area.

# ORCHESTRATING CHILD WINDOW CREATION:

## The frame window:

- The frame window plays a central role in creating child windows.
- It handles WM\_COMMAND messages that indicate menu selections for creating new windows.
- It prepares an MDICREATESTRUCT structure with important window information, such as the window title (commonly a filename in real-world applications), desired window styles (like scroll bars or minimized/maximized state), and any optional data to be shared with the child window.
- Finally, it sends a WM\_MDICREATE message to the client window, along with the MDICREATESTRUCT, to initiate the creation of the child window.

## Responsibilities of the Client Window:

### Message Handler:

The client window is responsible for **receiving the WM\_MDICREATE message** and promptly creating the child window based on the provided information.

### Updating the Submenu:

It **continuously updates the designated submenu** with the title of the newly created child window. This ensures a consistent user interface where the submenu reflects the available document windows.

### Efficient Document List Management:

The **client window effectively manages the document list** within the Window submenu. It automatically adds up to nine child window items, each assigned a numbered shortcut for quick navigation.

In cases where **more than nine windows are created**, the client window carefully includes a "More Windows" item. This item provides access to a comprehensive dialog that lists all active windows, allowing the user to navigate between them conveniently.

## Alternative Approaches and Data Exchange:

**Direct Creation:** While the frame window is typically responsible for creating child windows, developers can also directly create child windows using the `CreateMDIWindow` function. This provides flexibility in certain scenarios where the frame window's involvement may not be necessary.



**Data Sharing:** The `IParam` field of the `MDICREATESTRUCT` structure serves as a valuable mechanism for sharing data between the frame window and the child window during creation. This allows for efficient communication and coordination between the two, enabling features such as passing filenames or custom settings from the frame window to the child window.



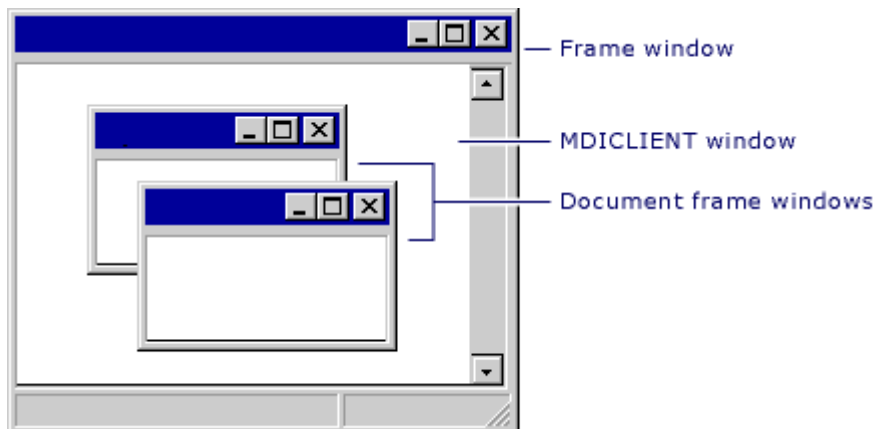
## Dynamic Menu Handling:

**Automatic Updates:** The client window takes care of automatically adding and removing child window items from the Window submenu as windows are created and destroyed. This dynamic behavior ensures that the menu always accurately reflects the current state of the application. It enhances user navigation and window management by providing an up-to-date list of available windows.



## Key Takeaways for WinAPI Developers:

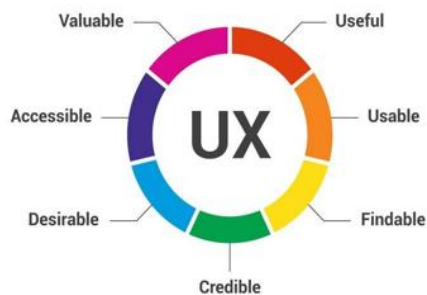
**MDICREATESTRUCT:** Familiarize yourself with this structure to effectively define the properties of child windows during their creation. Understanding its fields and their significance will help you customize and tailor the behavior of child windows according to your application's requirements.



**WM\_MDICREATE:** Comprehend the role of this message in facilitating communication between the frame window and the client window. It serves as a means to request the creation of child windows and allows for the exchange of relevant information.



**Client Window's Control:** Recognize the central role played by the client window in managing the document list and handling menu updates. Understanding this responsibility will enable you to create a seamless user experience with efficient window navigation and management.



**Data Sharing:** Take advantage of the lParam field in the MDICREATESTRUCT structure to establish efficient communication between the frame window and the child window. Utilizing this field for data exchange can enhance the functionality and customization of your application.





# FRAME WINDOW: THE ORCHESTRATOR OF MDI INTERACTIONS

Within an MDI application, the frame window serves as the **central hub for managing menu commands and coordinating actions** among its child windows. It gracefully handles user interactions with menus and ensures that messages are delivered to the appropriate destinations, fostering a seamless user experience.

## Menu Commands: A Variety of Actions

**Closing a Child Window:** The frame window initiates this process by:



- **Retrieving the handle** of the active child window using WM\_MDIGETACTIVE, ensuring focus on the intended window.
- **Respectfully requesting closure confirmation** from the child window via WM\_QUERYENDSESSION to avoid unexpected data loss.
- If the **child window grants permission**, the frame window decisively sends WM\_MDIDESTROY to the client window, empowering it to undertake the actual destruction of the window.
- **Exiting the Application:** The frame window initiates termination by sending a WM\_CLOSE message to itself. This message triggers a sequence of events that ultimately results in the graceful closure of all windows and the release of application resources.

**Arranging Child Windows:** The frame window offers users multiple ways to organize the workspace by sending specific messages to the client window:



- **WM\_MDITILE** for a tiled arrangement, neatly stacking windows side by side.
- **WM\_MDICASCADE** for a cascading arrangement, creating an overlapping, waterfall-like effect.
- **WM\_MDIICONARRANGE** for arranging minimized windows within the client area, maintaining visual order.
- **Closing All Child Windows:** This operation involves a choreographed dance between the frame window and child windows:
  - The **frame window** employs **EnumChildWindows** to meticulously iterate through each child window, inviting the **CloseEnumProc** function to the stage for processing.
  - **CloseEnumProc** diligently sends a **WM\_MDIESTORE** message to each child window, ensuring their visibility and readiness for closure.
  - It **then respectfully requests closure confirmation** via **WM\_QUERYENDSESSION**, respecting each child window's autonomy.
  - **Upon receiving a positive response**, **CloseEnumProc** decisively sends **WM\_MDIESTROY** to the client window, triggering the actual destruction of the willing child window.
- The **icon title window**, however, is gracefully exempt from this process, as indicated by a non-NULL return value from **GetWindow** with the **GW\_OWNER** argument.

## Message Forwarding: Empowering Child Windows

The **frame window** acknowledges the expertise of **child windows** in handling certain messages, particularly those pertaining to their unique functionalities.



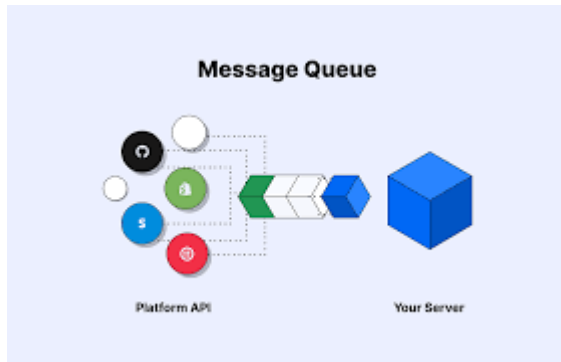
It **gracefully forwards unprocessed WM\_COMMAND messages** to the active child window, enabling them to respond to user interactions within their specific domains.



This image makes no sense, I just like looking at things while I read.

## DefFrameProc: The Unsung Hero of MDI

**Default Processing for Frame Windows:** This function diligently handles messages that the frame window procedure itself deems unnecessary for custom processing. It ensures that standard MDI behaviors are executed correctly, even when the frame window doesn't explicitly intervene.



**Essential for Certain Messages:** Specific messages must always be passed to DefFrameProc, even if the frame window traps them for other reasons:

- **WM\_MENUCHAR:** Handles keyboard navigation within menus, maintaining MDI consistency.
- **WM\_SETFOCUS:** Manages focus changes among windows, ensuring proper activation and deactivation.
- **WM\_SIZE:** Orchestrates window resizing behavior, coordinating with the client window to maintain a cohesive layout.
- **Unprocessed WM\_COMMAND messages:** Ensures that MDI-specific commands, such as those generated from the document list in the Window submenu, are handled appropriately.

## Document List Management:

- **Client Window's Responsibility:** The client window, not the frame window, bears the primary responsibility for managing the document list within the Window submenu. This includes:
- **Dynamically appending and removing child window items** as windows are created and destroyed.
- **Handling selections from the document list**, activating the corresponding child window.

## Window Handle Management:

**No Need for Explicit Lists:** The frame window elegantly avoids the need to maintain explicit lists of child window handles, streamlining its responsibilities.

**EnumChildWindows to the Rescue:** Whenever the frame window requires access to child window handles (such as during the Close All operation), it gracefully employs the EnumChildWindows function to effortlessly enumerate them on demand.



## Key Takeaways:

- ✓ **Frame Window as Coordinator:** The frame window plays a pivotal role in managing user interactions, coordinating actions among child windows, and forwarding messages to appropriate destinations.
- ✓ **DefFrameProc as a Reliable Partner:** This function ensures that essential MDI behaviors are handled correctly, even when custom message processing is involved.
- ✓ **Client Window's Role in Document Handling:** The client window manages the document list, providing a consistent user experience for navigating between multiple open documents.
- ✓ **Efficient Window Handle Management:** The frame window demonstrates a streamlined approach to window handle management, leveraging EnumChildWindows when needed, promoting a well-organized code structure.



