

# CHAPTER 19 — MULTIPLE DOCUMENT INTERFACE (MDI)

## What MDI Is (That's It)

**MDI (Multiple-Document Interface)** is a Windows application design where **one main window hosts multiple document windows** inside it.

Think:

- One app window
- Many document windows inside it
- All managed together

That's the core idea.

## Core Components

### Parent Window

- The main application window
- Owns menus, status bar, toolbar
- Contains a special **MDI client area**

### MDI Client Area

- A child window created with MDICLIENT
- Acts as a **workspace**
- All document windows live here

### Child Windows (Documents)

- Represent individual documents
- Look like mini windows:
  - Title bar
  - System menu
  - Min/Max/Close
- **No independent menus** — they use the parent's menu

## Active Document

- Only one child window is active at a time
- Shown with a highlighted title bar
- Receives keyboard input and commands

## How MDI Behaves

### Opening Documents

- Each document opens as a new MDI child window
- All remain inside the parent's workspace

### Arranging Windows

Users can:

- **Cascade** (stacked)
- **Tile** (horizontal / vertical)
- **Manually resize and move**

Handled through standard MDI commands.

### Switching Between Documents

- Mouse click
- Keyboard shortcuts (e.g. Ctrl+F6)
- Window menu selection

## Menu & UI Behavior (Important)

### Menu Sharing

- Only the **parent window owns menus**
- Menus adapt based on the **active child**
- Some items enable/disable depending on context

## Maximized Child Window

- Child's title merges into parent's title bar
- System buttons move into the parent frame
- Gives the illusion of a single unified window

## Minimized Child Window

- Shrinks into an icon inside the workspace
- Does **not** minimize to the taskbar

## Keyboard Shortcuts (Classic MDI)

ESSENTIAL WINDOWS / MDI SHORTCUTS	
SHORTCUT KEY	COMMAND ACTION & CONTEXT
Ctrl + F4	<b>Close Active Document:</b> Terminates the active MDI child window without exiting the app.
Alt + F4	<b>Close Application:</b> Sends WM_CLOSE to the main parent window (ends process).
Ctrl + F6	<b>Next Document:</b> Cycles focus through open MDI child windows.
Alt + Space	<b>Parent System Menu:</b> Opens the move/size/minimize menu for the main frame.
Alt + - (Minus)	<b>Active Child System Menu:</b> Opens the system menu for the specific active MDI document.

These are **built-in MDI conventions**, not app-specific tricks.

---

## The Window Menu

A dedicated menu (usually before *Help*) that:

- Lists all open documents
- Lets users activate a document
- Provides **Cascade / Tile** commands

This menu is automatically managed by Windows when using MDI correctly.

---

## Why MDI Was Popular

### Advantages

- Keeps related documents grouped
- Easy side-by-side comparison
- Avoids desktop clutter
- Centralized menus and commands

### Typical MDI Apps (Classic Era)

- Older Microsoft Word / Excel
- Adobe Photoshop
- IDEs and code editors

---

## Historical Context

- Introduced in **Windows 2.0**
- Painful to implement early on
- **Windows 3.0+** added real MDI support
- **Windows 98** further reduced developer effort with built-in behaviors.

MDI worked best when Windows *helped* instead of fighting you.

---

## Modern Reality

MDI is largely replaced by: **Tabbed interfaces** and **SDI (one document per window)**

Why?

- Simpler UX
- Better with modern window managers
- Easier cross-platform design

MDI still exists — just used more selectively.

---

## What Actually Matters for WinAPI Coders

### I. Programming Realities

- Parent + MDICLIENT + child windows
- Message routing is different
- Commands often forwarded to the active child
- Menu updates are **context-driven**

### II. Design Responsibility

- Windows gives structure
- **You decide usability**
- Poor MDI design = confusing app

---

### Summary (Keep This in Mind)

MDI applications use a parent window with an MDI client area to host multiple document windows. Only one document is active at a time, menus are shared and context-sensitive, and Windows provides built-in support for window arrangement, keyboard navigation, and menu integration. While powerful, MDI has largely been replaced by tabbed and SDI designs in modern applications.



# THE MDI ARCHITECTURE

MDI (Multiple-Document Interface) allows one application to open multiple documents at once (like older versions of Word or Excel). It is not just "windows inside a window." It is a specific three-layer hierarchy.

## The 3 Layers of MDI

To build an MDI app, you must create three specific types of windows in a specific order.

### I. The Frame Window (Grandparent)

This is the main application window.

- **Role:** The outer shell. It holds the main Title Bar, the System Menu, and the Border.
- **Style:** Standard `WS_OVERLAPPEDWINDOW`.
- **Behavior:** It runs the main Message Loop. It owns the Menu bar (because MDI child windows cannot have their own menus).

### II. The Client Window (Parent)

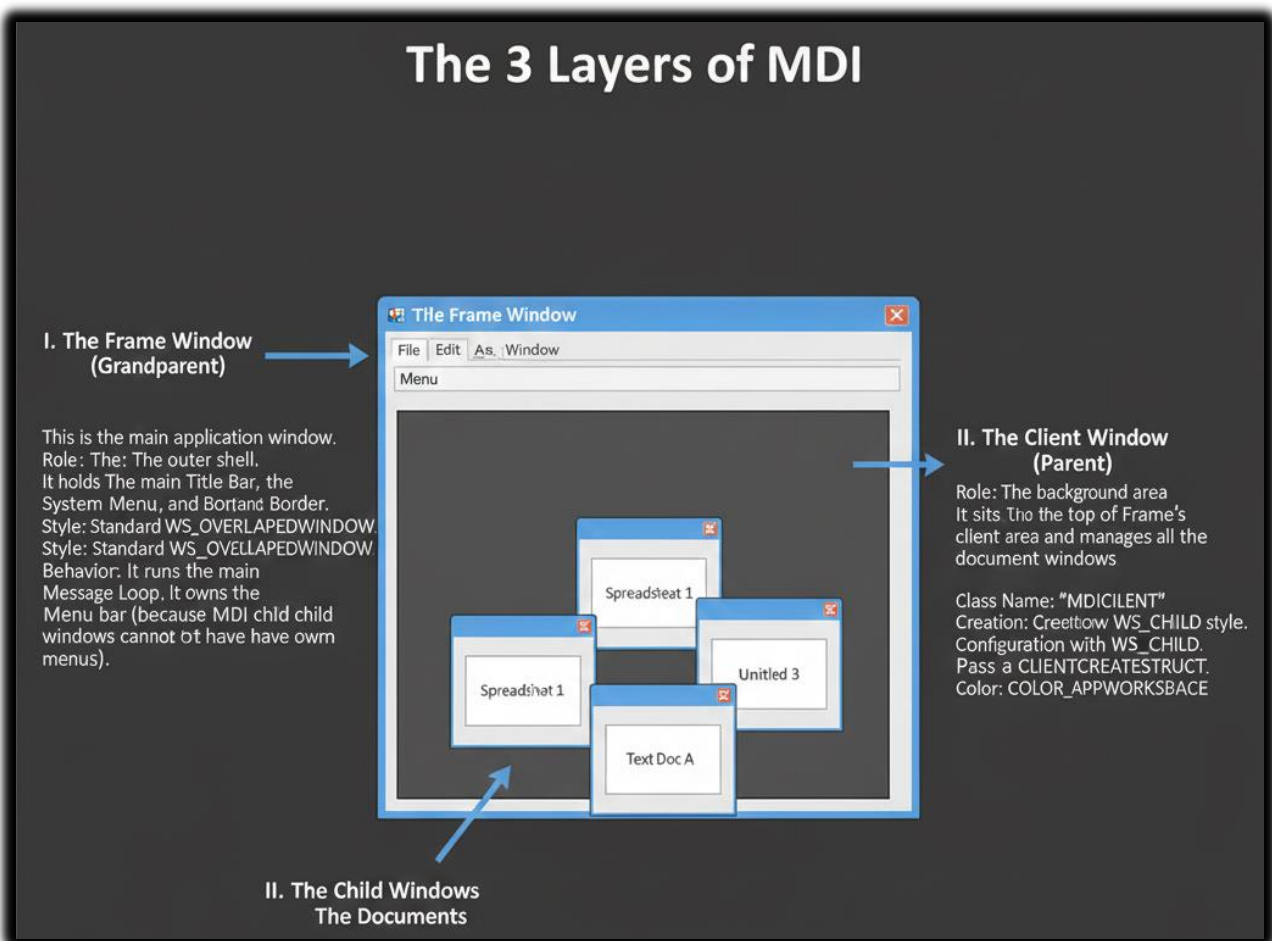
This is the background area (usually dark grey).

- **Role:** The Manager. It sits on top of the Frame's client area and manages all the document windows.
- **Class Name:** You do not create a custom class for this. You use the pre-defined system class "MDICLIENT".
- **Creation:** You call `CreateWindow` with the `WS_CHILD` style.
- **Configuration:** You pass a `CLIENTCREATESTRUCT` to tell it how to behave (e.g., which menu to use for the "Window" list).
- **Color:** It automatically uses the system color `COLOR_APPWORKSPACE`.

### III. The Child Windows (The Documents)

These are the actual document windows (Spreadsheet 1, Text Doc A) that float inside the workspace.

- **Role:** The content. They are children of the **Client Window**, not the Frame Window.
- **Creation:** You do not call `CreateWindow` directly. Instead, you fill out an `MDICREATESTRUCT` and send a `WM_MDICREATE` message to the **Client Window**. The Client Window then creates the child for you.
- **Restriction:** They are clipped to the Client Window. If you drag them outside the grey area, they disappear.

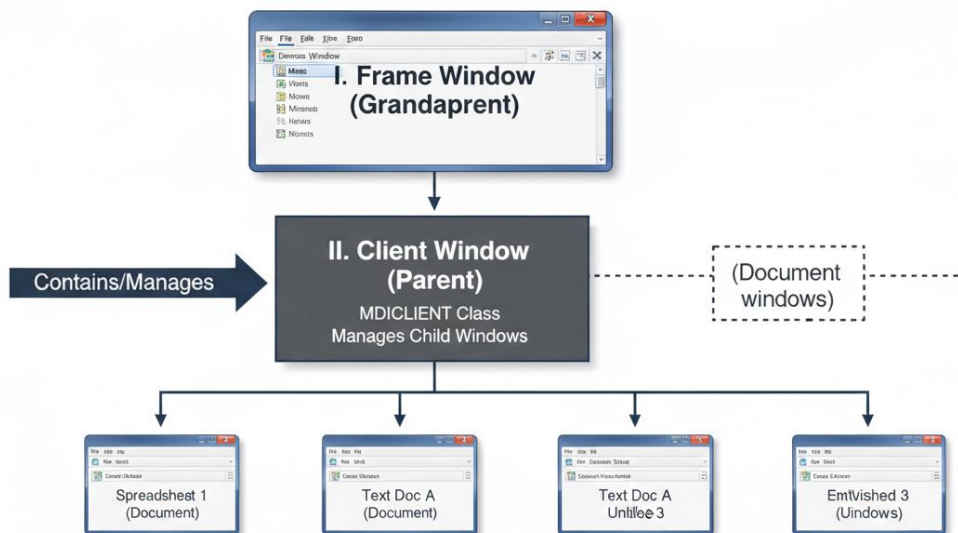


## The Hierarchy Cheat Sheet

It is vital to understand who owns whom:

1. **Frame Window** is the Parent of **Client Window**.
2. **Client Window** is the Parent of **Child Windows**.

**Why this matters:** If you send a message to the "Parent" of a Document Window, it goes to the **Client Window**, not the main Application Window. You often have to forward messages up the chain manually.



## MDI Development Implementation

We established the "Family Tree" (Frame to Client to Child). Now, let's look at the code required to make them communicate with each other.

MDI breaks some of the standard rules of Windows programming, so you must use specific functions to avoid breaking the application.

---

## The Three Pillars: Classes and Procedures

In a standard application, every window uses DefWindowProc.

In MDI, this is not the case.

### a) The Frame Window (The Boss)

Class:

You register your own custom window class.

Procedure:

You write your own window procedure.

Golden Rule:

Unhandled messages must be passed to DefFrameProc, not DefWindowProc.

Why?

DefFrameProc handles resizing the Client Window when the main window is resized and manages the menu bar when child windows are maximized.

### b) The Client Window (The Manager)

Class:

You use the predefined system class MDICLIENT.

Procedure:

None. You do not write a window procedure for this window. Windows manages it internally.

Job:

It sits between the Frame and the Child windows, managing the background, scrolling, and child window layout.

### c) The Child Windows (The Workers)

**Class:** You register your own custom class (for example, "MyDocumentClass").

**Procedure:** You write your own window procedure.

**Golden Rule:** Unhandled messages must be passed to DefMDIChildProc.

**Why?** DefMDIChildProc handles MDI-specific behavior, such as merging the child window menu into the Frame menu when the child is maximized.

---

## 2. The Structures: Birth Certificates

You cannot simply call `CreateWindow`.

MDI windows require extra data to know who they belong to.

### a) For the Client Window (`CLIENTCREATESTRUCT`)

When creating the Client Window, you pass a `CLIENTCREATESTRUCT`.

It tells the Client:

- `hWindowMenu`: The menu that lists open documents (for example, "1. Document A", "2. Document B")
- `idFirstChild`: The ID number for the first child window (usually starts at 50000)

### b) For the Child Windows (`MDICREATESTRUCT`)

When creating a new document, you fill out an `MDICREATESTRUCT`.

It tells the Client:

- `szClass`: The class name of the child window
- `szTitle`: The title shown in the child window's title bar
- `hOwner`: The application instance handle

---

## 3. The Message Traffic (How to Communicate)

In MDI, you do not directly control child windows.

Instead, you send commands to the Client Window and let it handle them.

### a) `WM_MDI` Messages

The Frame sends these messages to the Client Window:

- `WM_MDICREATE`: Create a new child window
- `WM_MDIACTIVATE`: Switch focus to another child window
- `WM_MDICASCADE`: Arrange all child windows in a cascade
- `WM_MDITILE`: Arrange child windows side by side

## b) The WM\_MDIACTIVATE Exception

This message works in two directions:

- Frame to Client: Request to activate a specific child window
  - Client to Child: Notification that the child gained or lost focus
- 

## 4. The Accelerator Trap (TranslateMDISysAccel)

MDI applications support special keyboard shortcuts:

- Ctrl+F6: Switch to the next document
- Ctrl+F4: Close the current document

Standard message handling does not process these shortcuts.

Inside the main message loop (the while(GetMessage...) loop), you must call TranslateMDISysAccel.

If you skip this step, users will not be able to switch documents using the keyboard.

---

## 5. Implementation Checklist

Follow this order when building an MDI application:

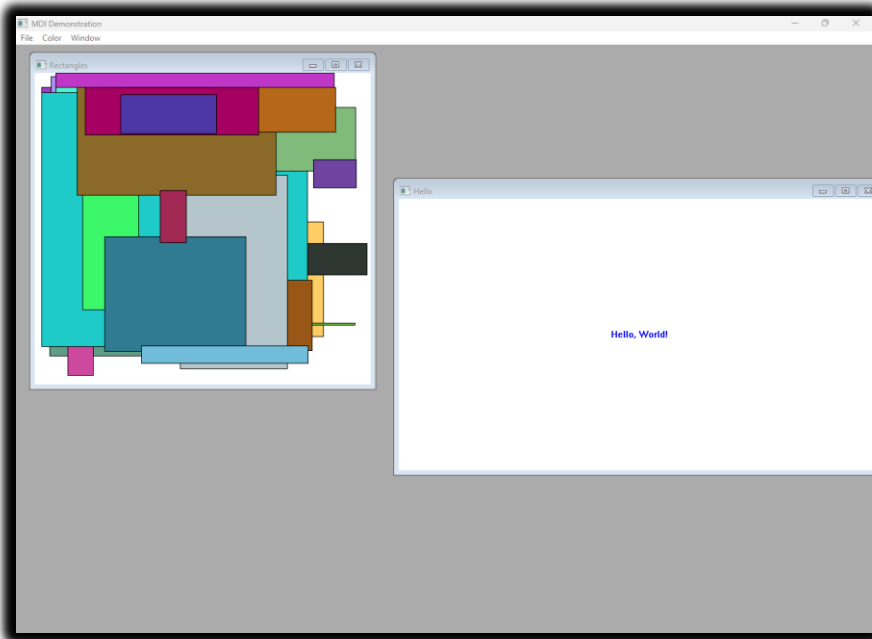
1. Register window classes for the Frame and Child windows
  2. Create the Frame window using CreateWindow
  3. Create the Client window inside the Frame's WM\_CREATE handler using the "MDICLIENT" class and pass CLIENTCREATESTRUCT
  4. Start the message loop and call TranslateMDISysAccel
  5. Create child windows by sending WM\_MDICREATE to the Client window
  6. Forward all unhandled messages to DefFrameProc or DefMDIChildProc
-

# MDIDEMO — WHAT THE PROGRAM IS REALLY SHOWING

**MDIDEMO** demonstrates a *complete, correct WinAPI MDI application*:

- One **frame window**
- One **MDI client window**
- Multiple **MDI child windows**
- Proper menu switching, message routing, accelerators, and cleanup

This is the *canonical* MDI pattern.



## Frame Window (FrameWndProc)

### I. Core Responsibilities

The frame window is the **brain** of the application.

It:

- Creates the **MDI client window** using the "MDICLIENT" class
- Owns **menus**, accelerators, and top-level commands
- Coordinates creation and destruction of child windows
- Routes commands to the active child when needed

## II. MDI Client Creation

- Uses CLIENTCREATESTRUCT
  - ✓ Specifies:
    - Which **Window submenu** will list documents
    - The base ID (IDM\_FIRSTCHILD) for child menu entries
- This enables Windows to auto-manage:
  - ✓ Child window lists
  - ✓ Window menu behavior
  - ✓ Keyboard shortcuts

## III. Message Routing

- Uses **DefFrameProc**
  - ✓ This is mandatory for proper MDI behavior
  - ✓ It handles:
    - Window menu updates
    - Activation changes
    - System MDI commands

## IV. Shutdown Handling

- On close or end session:
  - ✓ Attempts to close **all child windows**
  - ✓ Gives children a chance to confirm/save
  - ✓ Prevents data loss

## MDI Client Window (The Workspace)

The MDI client window:

- Receives WM\_MDICREATE / WM\_MDIDESTROY
- Creates child windows on behalf of the frame
- Manages the **Window submenu**
- Tracks:
  - ✓ Active child
  - ✓ Window order
  - ✓ Tiling / cascading

## Built-in MDI Messages

Handled automatically or via forwarding:

- WM\_MDIACTIVATE
- WM\_MDIGETACTIVE
- WM\_MDITILE
- WM\_MDICASCADE
- WM\_MDIICONARRANGE
- WM\_MDINEXT
- WM\_MDIMAXIMIZE

👉 This is why MDI works: **Windows does the heavy lifting.**

## Child Windows (Document Windows)

Each child window:

- Represents a document
- Has its own window procedure
- Relies on the **frame's menu**

## I. Child Window Lifecycle

- WM\_CREATE
  - ✓ Allocate per-window data
  - ✓ Store pointers using SetWindowLongPtr
- WM\_PAINT
  - ✓ Perform document-specific drawing
- WM\_COMMAND
  - ✓ Handle child-specific commands
- WM\_MDIACTIVATE
  - ✓ Enable/disable menu items based on focus
- WM\_CLOSE / WM\_QUERYENDSESSION
  - ✓ Ask for confirmation if needed
- WM\_DESTROY
  - ✓ Free allocated memory

## II. Message Handling

Uses **DefMDIChildProc**

- Required for proper MDI behavior
- Handles activation, sizing, system commands

## Memory Management (Why This Demo Is Solid)

- Per-window memory allocated with HeapAlloc
- Stored in window extra bytes
- Freed in WM\_DESTROY

This avoids:

- Globals
- Cross-window corruption
- Leaks

Clean WinAPI discipline.

## RectWndProc: Timers Done Right

- Uses SetTimer
- Periodically draws random rectangles
- Demonstrates:
  - ✓ Timers
  - ✓ Animation
  - ✓ Independent child behavior

This proves MDI children can be **fully autonomous**.

## Menus: The Hidden Complexity (Handled Correctly)

### Three Menu Templates

- **MdiMenuInit** → no documents
- **MdiMenuHello** → “Hello World” documents
- **MdiMenuRect** → rectangle documents

### Dynamic Menu Switching

- Menu changes based on **active child**
- Window submenu is always updated automatically
- Child windows don’t own menus — they influence them

## Key Constants

- INIT\_MENU\_POS, HELLO\_MENU\_POS, RECT\_MENU\_POS
- Tell Windows where the **Window submenu** lives
- IDM\_FIRSTCHILD ensures unique child menu IDs

## Accelerators & Message Loop (MDI-Specific)

Correct MDI message loop order:

1. TranslateMDISysAccel
2. TranslateAccelerator
3. TranslateMessage
4. DispatchMessage

This enables:

- Ctrl+F6
- Ctrl+F4
- Proper system navigation

Miss this → broken keyboard UX.

## Child Creation Flow (Real Signal)

1. User selects **New**
2. Frame prepares MDICREATESTRUCT
3. Frame sends WM\_MDICREATE to client
4. Client:
  - ✓ Creates child window
  - ✓ Adds it to Window submenu
  - ✓ Assigns numeric shortcut (up to 9)
5. Extra windows → **“More Windows...” dialog**

Zero manual bookkeeping required.

## Final Compression (What to Remember)

MDIDEMO shows the correct WinAPI MDI architecture: a frame window using DefFrameProc, an MDICLIENT workspace, child windows using DefMDIChildProc, structured creation via MDICREATESTRUCT, dynamic menus, accelerator translation, and disciplined memory management. Windows handles most MDI complexity when used correctly.

## Alternative Creation Paths

- **Standard way:**  
Frame window sends WM\_MDICREATE to the MDI client.
- **Shortcut:**  
CreateMDIWindow() can create a child directly.
- **Reality:**  
Both end up going through the **MDI client** anyway.  
The difference is *who initiates*, not *who controls*.

👉 Frame-first is clearer. Direct creation is situational.

## Data Exchange (Critical, Don't Forget This)

- MDICREATESTRUCT.lParam exists **for a reason**
- It's the clean channel to pass:
  - ✓ filenames
  - ✓ document state
  - ✓ configuration structs
- Child retrieves it in WM\_CREATE

**This is how real MDI apps avoid globals.**

## Dynamic Menu Handling (Let Windows Work)

- You **do not manually manage**:
  - ✓ child window lists
  - ✓ numbering
  - ✓ “More Windows...” dialog
- The **MDI client** handles:
  - ✓ adding/removing menu items
  - ✓ updating the Window submenu
  - ✓ tracking active documents

Your job: 🖱️ tell Windows *where* the Window submenu is 🖱️ let it do the rest

## The Three Things You Must Know Cold

### 1. MDICREATESTRUCT

Defines *what* gets created:

- title
- styles
- position
- startup state
- **lParam for data**

If you understand this struct, you understand MDI creation.

---

### 2. WM\_MDICREATE

Defines *how* creation is requested:

- Frame → Client
- Client → Child
- Centralizes lifecycle control

This is the handshake.

---

### 3. The Client Window Is the Boss

- Owns the document list
- Owns window arrangement
- Owns activation state
- Owns menu updates

The frame **coordinates**,  
the client **manages**,  
the child **does work**.

That's the hierarchy.

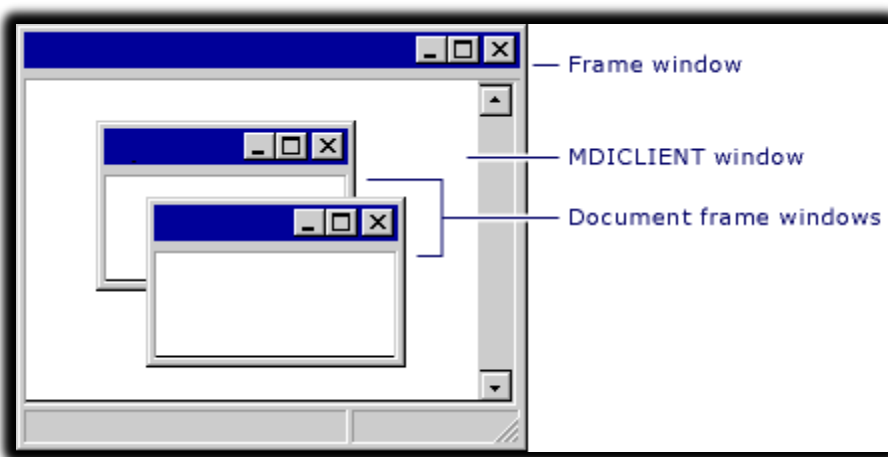
---

### Final One-Paragraph Doctrine (Write This in Your Head)

In WinAPI MDI applications, the frame window coordinates document creation, the MDI client window manages child window lifecycles and menus, and child windows focus solely on document behavior.

MDICREATESTRUCT defines creation parameters, WM\_MDICREATE initiates child creation, and the lParam field enables clean data exchange.

When used correctly, Windows automatically handles menu updates, window lists, activation, and keyboard navigation, allowing developers to focus on application logic instead of UI bookkeeping.

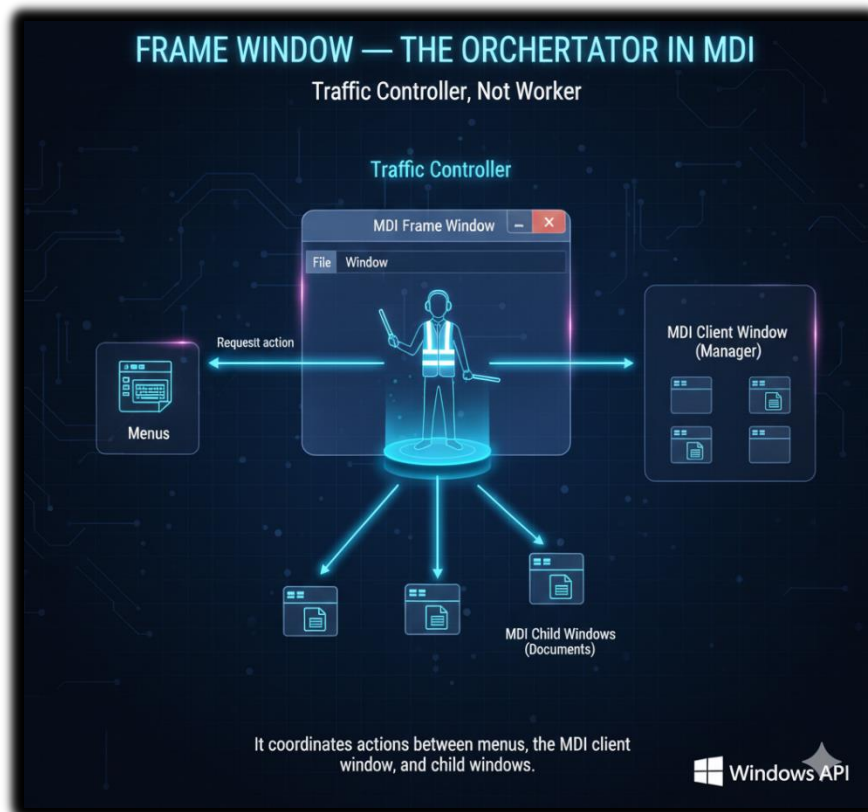


## FRAME WINDOW — THE ORCHESTRATOR IN MDI

The **frame window** is the command center of an MDI application.

It does **not** manage documents directly — it **coordinates** actions between menus, the MDI client window, and child windows.

Think: **traffic controller**, not worker.



### What the Frame Window Actually Does

#### I. Menu Command Coordination

The frame window receives menu input and decides **where it should go**.

Typical responsibilities:

- Closing the active child window
- Arranging child windows (tile, cascade, icons)
- Exiting the application
- Forwarding commands it doesn't own

It **initiates actions**, but the **MDI client executes them**.

## II. Closing a Child Window (Correct Flow)

1. Ask the MDI client for the active child (WM\_MDIGETACTIVE)
2. Ask the child if it agrees to close (WM\_QUERYENDSESSION)
3. If approved:
  - ✓ Send WM\_MDIDESTROY **to the client**, not directly to the child

👉 The client window owns child destruction.

## III. Exiting the Application

- The frame window sends WM\_CLOSE to itself
- Windows handles the shutdown cascade:
  - ✓ children → client → frame
- Clean, predictable teardown

## Arranging Child Windows

The frame window **does not rearrange windows itself**.

Instead, it sends commands to the client:

- WM\_MDITILE
- WM\_MDICASCADE
- WM\_MDIICONARRANGE

The client handles layout logic.

## Closing *All* Child Windows

Used sparingly, but important.

Process:

- Enumerate children with EnumChildWindows
- For each child:
  - ✓ Restore if minimized (WM\_MDIRESTORE)
  - ✓ Ask permission to close (WM\_QUERYENDSESSION)
  - ✓ Destroy via WM\_MDIDESTROY (sent to client)

Special case:

- Icon title windows are skipped (GW\_OWNER check)

No child list is stored.

Enumeration is done **on demand**.

## Message Forwarding (Critical Concept)

The frame window **does not steal commands** from children.

If it receives a WM\_COMMAND it doesn't handle:

- Forward it to the active child window

This keeps:

- logic local
- behavior modular
- menus context-aware

## DefFrameProc — Mandatory, Not Optional

DefFrameProc handles **core MDI behavior**.

Always pass these through:

- WM\_MENUCHAR
- WM\_SETFOCUS
- WM\_SIZE
- Unhandled WM\_COMMAND

If you don't, MDI **breaks in subtle ways**.

## Document List Reality Check

- The **client window** manages:
  - ✓ the Window submenu
  - ✓ document numbering
  - ✓ activation
- The frame window:
  - ✓ only tells the client *where* the Window menu is

Never manage document lists manually.

## Window Handle Management

- No stored arrays
- No global lists
- Use EnumChildWindows when needed

This keeps the frame window **stateless and clean**.

- **Frame window** → coordinates & routes
- **MDI client** → owns children & menus
- **Child windows** → do document work
- **DefFrameProc** → enforces MDI rules

If you remember this, you won't fight MDI ever again.

MDI Child windows are not just standard windows. They have three unique requirements: they need their own private memory, they need to control the main menu, and they have strict message-handling rules.

## I. Private Data Storage (The "Individuality" Problem)

In a standard app, you might use global variables to store the text color. In MDI, you can have 10 different windows open, each with a different color. You cannot use global variables because changing one window would change them all.

**The Solution:** You must attach the data structure **directly to the window handle**.

1. **Allocate Space:** When registering the child class (WNDCLASS), set `cbWndExtra` to the size of a pointer (e.g., `sizeof(void*)`).
2. **Attach Data:** In `WM_CREATE`, allocate a memory block (struct) for that specific document. Use `SetWindowLong` to save the pointer into the window's extra memory.
3. **Retrieve Data:** In every other message (like `WM_PAINT`), use `GetWindowLong` to retrieve that pointer so you know which color *this* specific window uses.



## II. Menu Swapping (The Chameleon Menu)

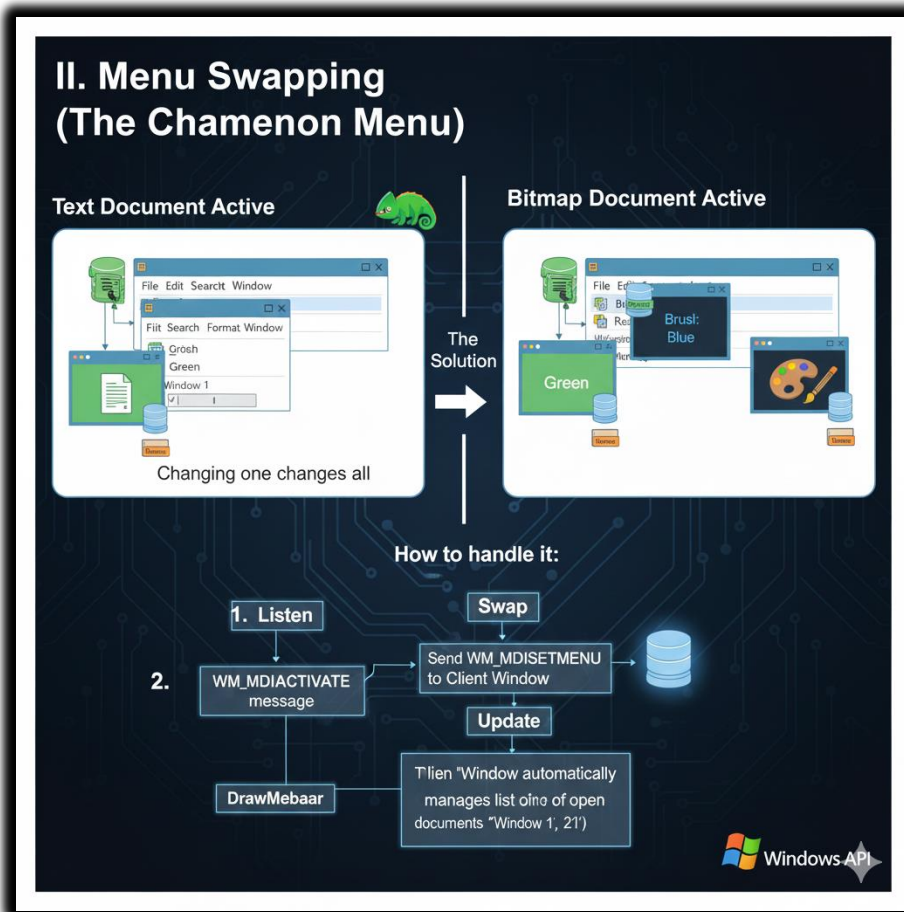
The Frame Window owns the Menu Bar, but the Child Window controls what is on it.

- **Text Document Active:** Menu shows "Edit, Search, Format."
- **Bitmap Document Active:** Menu shows "Brush, Color, Resize."

**How to handle it:**

1. **Listen:** Catch the WM\_MDIACTIVATE message. This tells you when your window gains or loses focus.
2. **Swap:** Send WM\_MDISETMENU to the Client Window. Tell it which Menu Handle to display.
3. **Update:** Call DrawMenuBar to refresh the visual immediately.

**The "Window" Menu List:** The Client Window automatically manages the list of open documents (Window 1, Window 2) at the bottom of the "Window" menu. You don't need to code this; just ensure your menu setup passes the correct handle to CLIENTCREATESTRUCT.



Just trial images.

### III. The Dangerous Exception (DefMDIChildProc)

This is the most common bug in MDI programming.

**Standard Rule:** If you handle a message (like WM\_SIZE), you usually return 0 and stop.

**MDI Rule:** You **must** pass certain messages to DefMDIChildProc **even if you handled them**.

If you block these messages, the MDI system breaks (e.g., minimizing a child window won't work correctly):

- WM\_CHILDACTIVATE
- WM\_GETMINMAXINFO
- WM\_MENUCHAR
- WM\_MOVE
- WM\_SETFOCUS
- WM\_SIZE
- WM\_SYSCOMMAND



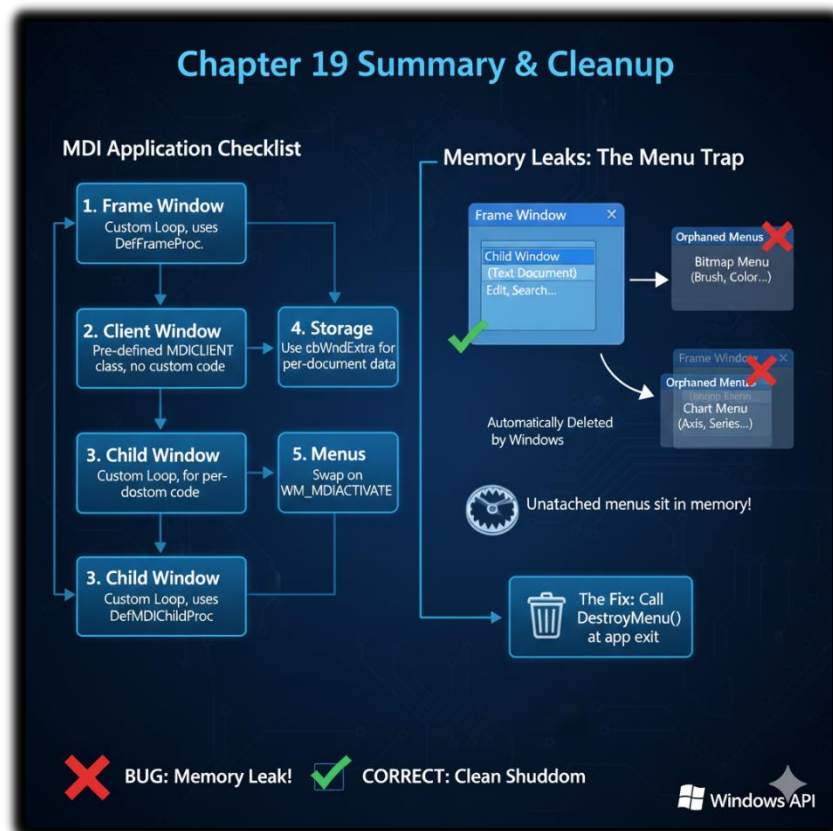
## IV. Cleanup and Memory Leaks

Because you are swapping menus, you have a resource danger.

**The Rule:** Windows only automatically deletes menus that are **currently attached** to a window when that window closes.

**The Trap:** If you have 3 different menu types (Text, Bitmap, Chart) and you close the app, Windows deletes the *current* one. The other two are sitting in memory, unattached.

**The Fix:** At the very end of WinMain, before return, you must manually call DestroyMenu for any menu handles that are not currently active.



### Chapter 19 Summary checklist

- 1. Frame Window:** Custom Loop, uses DefFrameProc.
- 2. Client Window:** Pre-defined MDIClient class, no custom code.
- 3. Child Window:** Custom Loop, uses DefMDIChildProc.
- 4. Storage:** Use cbWndExtra to keep data separate for each document.
- 5. Menus:** Swap them on WM\_MDIACTIVATE.