MDI – MultipleDocument Interface – Interfejs obsługi wielu dokumentów.

W programie MDI obszar roboczy głównego okna aplikacji nie jest wykorzystywany bezpośrednio do wyświetlania danych wyjściowych programu. Obszar ten może zawierać wiele okien potomnych, z których każde będzie zawierać dokument. Takie okna potomne przypominają normalne okna aplikacji, z tym że, do okien dokumentu stosuje się menu okna głównego aplikacji.

Główne okno aplikacji nazywamy „framugą" (frame window) i jest to okno w stylu WS_OVERLAPPEDWINDOW.

W WinMain programu MDI:

- rejestrujemy klasę framugi, następnie rejestrujemy klasy okien dokumentów. Dla okien potomnych nie przypisujemy menu, dla framugi można podać teraz lub przy tworzeniu okna. Dla okien potomnych możemy przydzielić więcej pamięci podając: `cbWndExtra!=0`. Każda klasa ma własną procedurę okna.
- przy pomocy *LoadMenu* pobieramy uchwyty do menu odpowiadającym: brak dokumentów i aktywne dokumenty danego rodzaju.
- przy pomocy *GetSubMenu* pobieramy uchwytu do podmenu, które będzie zawierać odwołania do otwartych dokumentów. Przy braku dokumentów nie posiadamy menu „Okno", mimo to podajemy wartość 0.
- jeżeli istnieje ładujemy tablicę klawiszy skrótów.
- za pomocą *CreateWindow* tworzymy framugę.
- Podczas przetwarzania WM_CREATE framuga tworzy okno klienta typu WS_CHILD, korzystając z predefiniowanej klasy „MDICLIENT". Okno klienta pokrywa obszar roboczy framugi i jest odpowiedzialne za obsługę MDI. Kolor okna klienta jest systemowy COLOR_APPWORKSPACE. Ostatnim parametrem funkcji *CreateWindow* jest wskaźnik na strukturę **CLIENTCREATESTRUCT**. Struktura ta składa się z dwóch pól:
  `hWindowMenu` – uchwyt podmenu do którego zostanie dołączona lista dokumentów.
  `idFirstChild` – identyfikator menu, który ma zostać skojarzony z oknem pierwszego dokumentu na liście dokumentów. Identyfikator ten powinien mieć wartość większą niż wszystkie inne identyfikatory menu.
- w WinMain pobieramy uchwyt do okna klienta za pomocą `GetWindow(hwndFrame, GW_CHILD)`
- pokazujemy i odświeżamy framugę.
- tworzymy pętle komunikatów:

```
while (GetMessage (&msg, NULL, 0, 0))
{
  if (!TranslateMDISysAccel (hwndClient, &msg)
     && !TranslateAccelerator (hwndFrame, hAccel, &msg))
  {
    TranslateMessage (&msg) ;
    DispatchMessage (&msg) ;
   }
}
```

  Funkcja *TranslateMDISysAccel* tłumaczy naciśnięcia klawiszy, które mogą odpowiadać specjalnym klawiszom skrótu MDI.
- niszczymy menu dokumentów.

W procedurze okna framugi przede wszystkim przetwarzamy komunikaty menu. Wśród nich powinno być polecenie otwarcia nowego okna dokumentu.
Okna dokumentów tworzymy przez zainicjowanie struktury MDICREATESTRUCT i wysłanie do okna-klienta komunikatu WM_MDICREATE ze wskaźnikiem na strukturę.

```
Typedef struct tagMDICREATESTRUCT {
LPCTSTR szClass;      // nazwa klasy okna dokumentu
LPCTSTR szTitle;      // napis na pasku tytułowym
HANDLE hOwner;        // uchwyt wystąpienia do programu w ramach którego tworzone
                      // są dokumenty
int x;
int y;
int cx;
int cy;
DWORD style           // WS_MINIMIZE (zminimalizowane), WS_MAXIMIZE,
                      // WS_HSCROLL, WS_VSCROLL
LPARAM lParam;        // umożliwia framudze i oknu dok. dzielenie rzeczy. Składowa ta
                      // z reguły zawiera wskaźnik na strukturę.
} MDICREATESTRUCT, *LPMDICREATESTRUCT;
hwndChild =
     =(HWND)SendMessage( hwndClient,WM_MDICREATE,0,
                         (LPARAM)(LPMDICREATESTRUCT)&mdicreate);
```

Obsługa innych komunikatów framugi:
Zamknięcie dokumentu:
Pobieramy uchwyt aktywnego okna dokumentu:

```
hwndChild = (HWND) SendMessage (hwndClient, WM_MDIGETACTIVE,
                                    0, 0) ;
```

Jeżeli okno potomne odpowie twierdząco na komunikat WM_ QUERYENDSESSION wysyłamy do okna klienta komunikat, który spowoduje zamknięcie okna potomnego (WM_MDIDESTROY)

```
if (SendMessage (hwndChild, WM_QUERYENDSESSION, 0, 0))
 SendMessage (hwndClient, WM_MDIDESTROY,(WPARAM) hwndChild,0);
```

Wyjście z programu:
Wysyłamy do okna framugi (do siebie) komunikat WM_CLOSE

```
SendMessage (hwnd, WM_CLOSE, 0, 0);
```

Aby odpowiednio ustawić okna dokumentów należy wysłać do okna-klienta komunikaty:
WM_MDITILE, WM_MDICASCADE, WM_MDIICONARRANGE.

Zamknij wszystkie dokumenty:
Wywołujemy funkcje, która wysyła uchwyt kolejnych okien potomnych danego rodzica do podanej funkcji CALLBACK. Funkcja kontynuuje wykonanie dopóki nie skończą się okna potomne lub dopóki funkcja nie zwróci 0:

```
BOOL EnumChildWindow(HWND hwndParent, WNDENUMPROC lpEnumFunc,
                     LPARAM lParam)
```

Funkcja typu CALLBACK ma następującą postać:

```
BOOL CALLBACK CloseEnumProc (HWND hwnd, LPARAM lParam);
```

W ramach tej funkcji dokonujemy przywrócenia poprzedniej wielkości i pozycji okna dokumentu jeżeli było zminimalizowane lub zmaksymalizowane wysyłając komunikat do okna-klienta WM_MDIRESTORE, którego wParam odpowiada uchwytowi okna potomnego.
```
SendMessage (GetParent(hwnd),WM_MDIRESTORE, (WPARAM) hwnd, 0);
```
Następnie jeżeli okno potomne odpowie twierdząco na komunikat
WM_ QUERYENDSESSION wysyłamy do okna klienta komunikat WM_MDIDESTROY.

Pozycje menu, które chcemy aby były obsłużone przez aktywne okno dokumentu:
```
default :
// pobieramy uchwyt aktywnego okna dokumentu
if (IsWindow (hwndChild)) // sprawdzamy czy takie okno istnieje
   SendMessage (hwndChild, WM_COMMAND, wParam, lParam);
```

Wszystkie komunikaty, które nie obsługujemy w procedurze okna framugi, oraz te które przekazujemy do aktywnego okna dokumentu muszą trafić do funkcji *DefFrameProc*:
```
return DefFrameProc(hwnd, hwndClient, iMsg, wParam, lParam);
```

Komunikaty okien dokumentów:
WM_CREATE:
Możemy zdefiniować dane przechowywane tylko dla konkretnego okna:
Przy pomocy funkcji `HeapAlloc` przydzielamy nie zmienialny obszar pamięci na stosie.
```
LPVOID HeapAlloc(HANDLE hHeap, DWORD flag, SIZE_T dwByte);
```
hHeap – uchwyt stosu który otrzymujemy za pomocą funkcji:
`HANDLE GetProcessHeap()` – funkcja zwraca uchwyt stosu wywołującego tę funkcję procesu.
Flag – specyfikuje opcje przydziału pamięci. W naszym przypadku używamy:
HEAP_ZERO_MEMORY (wskazuje, że pamięć jest inicjalizowana od 0)
dwByte – ilość bajtów do zaalokowania;
Funkcja zwraca wskaźnik na blok przydzielonej pamięci.
Zapisujemy wskaźnik na przydzieloną pamięć za pomocą
```
SetWindowLong(hwnd, 0, (long) lpData)
```
gdzie trzeci parametr odpowiada wskaźnikowi na pamięć.
Możemy później pobrać ten wskaźnik za pomocą funkcji:
```
lpData = (LPDATA) GetWindowLong(hwnd, 0) ;
```

WM_MDIACTIVATE :
Komunikat wysyłany gdy okno staje się nieaktywne lub aktywne.
lParam - zawiera uchwyt okna aktywnego
Jeżeli okno dokumentu staje się aktywne możemy zmienić menu wysyłając komunikat WM_MDISETMENU:
```
SendMessage(hwndClient,WM_MDISETMENU,(WPARAM)hMenuDocument,
            (LPARAM)hMenuDocWindow);
```
Ostatni parametr wskazuje na uchwyt podmenu, do którego należy wpisać odwołania do otwartych dokumentów.
Jeżeli okno przestaje być aktywne ustawiamy na menu bez dokumentów.
Następnie wywołujemy funkcje która odrysuje menu:
```
DrawMenuBar(hwndFrame);
```

WM_DESTROY:
Jeżeli w WM_CREATE przydzieliliśmy pamięć to należy ją tutaj zwolnić za pomocą funkcji:
```
HeapFree(GetProcessHeap(), 0, lpData) ;
```

Wszystkie nie obsłużone komunikaty przekazujemy do:
```
DefMDIChildProc(hwnd, iMsg, wParam, lParam);
```
Przekazujemy tam także, niezależnie od tego czy je obsłużymy komunikaty:
WM_GETMINMAXINFO, WM_MENUCHAR, WM_MOVE, WM_SETFOCUS,
WM_SIZE, WM_SYSCOMMAND.