Komunikaty mogą być:

- kolejkowane – komunikaty, które Windows umieszcza w kolejce komunikatów, skąd są pobierane w pętli komunikatów.
- niekolejkowane – wysyłane bezpośrednio do procedury okna.

Standardowe komunikaty kolejkowane:

- od klawiatury (np. `WM_KEYDOWN, WM_KEYUP, WM_CHAR`)
- od myszy (`WM_MOUSEMOVE, WM_LBUTTONDOWN`)
- zegar systemu (`WM_TIMER`)
- odświeżanie ekranu (`WM_PAINT`)
- zakończenie działania programu (`WM_QUIT`)

Pozostałe nie są kolejkowane (np. `WM_CREATE, WM_MOVE, WM_SHOWWINDOW`).

Funkcje wysyłające komunikaty (mają takie same parametry jak procedura okna):

- funkcja wstawiająca komunikat do kolejki komunikatów, funkcja powraca nie czekając na przetworzenie komunikatu
  ```
  BOOL PostMessage(HWND hWnd, UINT Msg, WPARAM wParam,
                   LPARAM lParam)
  ```
- funkcja wysyłająca komunikat do okna (z ominięciem kolejki), funkcja czeka na zakończenie przetwarzania komunikatu
  ```
  LRESULT SendMessage(HWND hWnd, UINT Msg, WPARAM wParam,
                      LPARAM lParam)
  ```

W Windows mamy dużo „czasu martwego” tzn. czasu kiedy kolejka komunikatów jest pusta. Możemy w naszym programie przechwycić sterowanie w tym martwym czasie za pomocą funkcji *PeekMessage*.

```
BOOL PeekMessage(LPMSG lpMsg, HWND hWnd, UINT wMsgFilterMin,
                 UINT wMsgFilterMax, UINT wRemoveMsg)
```

- funkcja działa podobnie jak *GetMessage*. Różnica polega na tym że:

- *PeekMessage* zwraca TRUE jeżeli kolejka zwiera jakiś komunikat, FALSE w przeciwnym wypadku.
- *PeekMessage* może nie usunąć komunikatu z kolejki jeżeli
  wRemoveMsg = PM_NOREMOVE.
  Usunie go jeżeli:
  wRemoveMsg = PM_REMOVE.
- *GetMessage* nie zwraca sterowania do programu jeżeli nie pobierze komunikatu, natomiast *PeekMessage* zawsze kończy działanie bez względu na ilość komunikatów w kolejce.

Pętla komunikatów będzie miała następującą postać:

```
while(TRUE)
{    if(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
     {
     if(msg.message == WM_QUIT)
          break;
     TranslateMessage(&msg);
     DispatchMessage(&msg);
     }
     else {     // działania wykonane między kolejnymi sprawdzeniami kolejki   }
}
```

Copyright © 2001-2003 Ewa Lach

Obsługa wybranych komunikatów:

`WM_PAINT` – informuje program, że część lub całość obszaru okna roboczego uległa ‘unieważnieniu’ i wymaga odświeżenia.

`BOOL` **`UpdateWindow`**`(HWND hWnd)` **–** funkcja wymusza odświeżenie zawartości okna.

W trakcie przetwarzania komunikatu powinno nastąpić zatwierdzenie unieważnionego obszaru. Dokonuje się tego za pomocą *BeginPaint* lub *ValidateRect.*

Malując obszar roboczy twojego okna korzystasz z windowsowych funkcji GDI. Niemal każda funkcja GDI wymaga podania wartości uchwytu kontekstu urządzenia.
Kontekst urządzenia (DC – device context) jest zwykłą strukturą zakładaną i zarządzaną przez GDI. Kontekst urządzenia jest związany z konkretnym urządzeniem, takim jak drukarka, ploter czy ekran. W przypadku ekranu kontekst urządzenia jest związany z konkretnym oknem.

Program nie może nic wykreślić, dopóki nie otrzyma uchwytu kontekstu urządzenia. Po zakończeniu kreślenia program powinien zwolnić ten uchwyt.

Większość aplikacji windowsowych stosuje jedną z dwóch metod w celu otrzymania uchwytu kontekstu urządzenia.

Pierwszej metody używamy w trakcie przetwarzania komunikatu WM_PAINT.
W najbardziej typowym przypadku przetwarzanie komunikatu wygląda następująco:

```
case WM_PAINT:
     PAINTSTRUCT ps;
     HDC hdc;
     hdc = BeginPaint(hwnd, &ps);
     [wywołania funkcji GDI]
     EndPaint(hwnd, &ps);
     return 0;
```

`PAINTSTRUCT` – struktura informacji o malowaniu, zakładana przez Windows dla każdego okna.

```
typedef struct tagPAINTSTRUCT
{
  HDC  hdc;           \\ uchwyt kontekstu urządzenia
  BOOL fErase;        \\ True oznacza, że Windows wymazał tło unieważnionego
                      \\ prostokąta przy pomocy pędzla określonego w polu
                      \\ hbrBackground klasy okna.
  RECT rcPaint;       \\ Wyznacza granicę unieważnionego prostokąta. Wymiary
                      \\ wyrażone są w pikselach względem lewego górnego rogu
                      \\ obszaru roboczego.
  BOOL fRestore;      \\ pole wykorzystywane wyłącznie przez Windows
  BOOL fIncUpdate;    \\ j. w
  BYTE rgbReserved;   \\ j. w.
} PAINTSTRUCT;
```

Windows wypełnia jej pola podczas wywołania funkcji *BeginPaint.*
`RECT` – struktura składająca się z czterech pól: `left, top, right, bottom`.

Copyright © 2001-2003 Ewa Lach

`HDC` **`BeginPaint`**`(HWND hwnd, LPPAINTSTRUCT lpPaint)`
– funkcja czyści tło obszaru roboczego, wypełnia pola struktury `PAINTSTRUCT`, zatwierdza unieważniony obszar. Zwraca uchwyt kontekstu urządzenia dla odświeżanego obszaru lub NULL w przypadku błędu.
Każdemu wywołaniu funkcji `BeginPaint` musi towarzyszyć wywołanie funkcji EndPaint.
`BOOL` **`EndPaint`**`(HWND hwnd, LPPAINTSTRUCT lpPaint)`
– funkcja zwalnia uchwyt kontekstu urządzenia.

Korzystając z kontekstu urządzenia zwróconego przez *`BeginPaint`* możemy malować jedynie w unieważnionym obszarze. Aby malować na zewnątrz prostokąta *`rcPaint`* należy wywołać funkcję *`InvalidateRect(hWnd,NULL,TRUE)`* przed wywołaniem *`BeginPaint`*.

`BOOL` **`InvalidateRect`**`(HWND hWnd, CONST RECT *lpRect,BOOL bErase)`
– funkcja unieważnia zadany region.
`hWnd` - uchwyt okna, którego obszar ma zostać unieważniony. Jeżeli = NULL wszystkie okna są unieważniane.
`lpRect` - wskaźnik na prostokąt który ma być dodany do obszaru do odświeżenia. Jeżeli = NULL cały obszar roboczy zostanie odświeżony.
`bErase` - wskazuje czy ma zostać wymazane tło unieważnionego prostokąta.

Możemy pobrać współrzędne unieważnionego regionu za pomocą funkcji:
`BOOL` **`GetUpdateRect`**`(HWND hWnd ,CONST RECT *lpRect, BOOL bErase)`

Przeciwną funkcją do *`InvalidateRect`* jest funkcja *`ValidateRect`* która zatwierdza unieważniony region.
`BOOL` **`ValidateRect`**`(HWND hWnd , CONST RECT * lpRect)`

Uchwyt kontekstu urządzenia możemy także otrzymać jeżeli chcemy kreślić w obszarze roboczym przy okazji obsługi innych komunikatów niż WM_PAINT. Wykorzystujemy do tego funkcje **GetDC**, która zwraca uchwyt kontekstu urządzenia. Przy pomocy funkcji **ReleaseDC** zwalniamy uchwyt.

```
hdc = GetDC(hwnd);
[ wywołania funkcji GDI ]
ReleaseDC(hwnd,hdc)
```

Różnice między *`GetDC`* i *`BeginPaint`*: - *`GetDC`* zwraca uchwyt kontekstu urządzenia odnoszący się do całego obszaru roboczego i nie zatwierdza żadnego unieważnionego regionu.

Funkcja GDI, wyświetlającą tekst w obszarze roboczym :

```
BOOL TextOut( HDC hdc, int nXStart, int nYStart,
              LPCTSTR lpString, int cbString )
```

`hdc` – uchwyt kontekstu urządzenia
`nXStart` – punkt początkowy napisu na osi poziomej określany względem lewego górnego rogu obszaru roboczego
`nYStart` - punkt początkowy napisu na osi pionowej określany względem lewego górnego rogu obszaru roboczego
`lpString` – wskaźnik do ciągu znaków.
`cbString` – liczba znaków w ciągu.

Copyright © 2001-2003 Ewa Lach

WM_CREATE – to pierwszy komunikat który trafia do procedury okna. Generuje go Windows przy okazji wywołania funkcji *CreateWindows*.

WM_CLOSE – komunikat ten oznacza, że okno powinno zostać zamknięte. Jeżeli procedura okna przekaże ten komunikat do *DefWindowProc* to funkcja ta odpowie na ten komunikat wywołując funkcję *DestroyWindow*, która sprawi, że Windows wyśle do okna komunikat WM_DESTROY. Jeżeli sami obsłużymy ten komunikat to możemy powstrzymać Windows przed zamknięciem okna.

Obsługa komunikatów klawiatury.

W chwili wciśnięcia lub zwolnienia klawisza komunikat na ten temat jest przesyłany do kolejki komunikatów systemu. Następnie Windows przesyła po kolei komunikaty do kolejki komunikatów programu który posiada tzw. „ognisko wejścia" (fokus). W kolejnym etapie komunikat jest kierowany do procedury okna.
Dwuetapowość związana jest z sytuacją gdy użytkownik szybciej uderza w klawisze niż program jest w stanie przetworzyć nadchodzące komunikaty. Jeżeli jeden z komunikatów przekazuje ognisko innej aplikacji, Windows może wysyłać kolejne komunikaty nowej procedurze okna.
Okno posiadające „ognisko wejścia" jest albo oknem aktywnym, albo oknem potomnym okna aktywnego. Samo okno potomne nigdy nie jest oknem aktywnym. Jeżeli okno aktywne zostanie zmniejszone do ikony, to żadne okno nie ma ogniska.

WM_SETFOCUS – komunikat który informuje o tym, że okno właśnie otrzymuje ognisko.
wParam – uchwyt okna tracącego ognisko.

WM_KILLFOCUS – sygnalizuje utratę ogniska przez okno.
wParam – uchwyt okna otrzymującego ognisko.

Komunikaty związane z klawiaturą dzielimy na dwie grupy:
- Komunikaty znakowe
- Komunikaty związane z naciśnięciem klawisza

Komunikaty związane z naciśnięciem klawisza:

	Klawisz wciśnięty	*Klawisz zwolniony*
Klawisz zwykły (bez [Alt])	**WM_KEYDOWN**	**WM_KEYUP**
Klawisz systemowy (klawisz naciśnięty razem z [Alt])	**WM_SYSKEYDOWN**	**WM_SYSKEYUP**

lParam – zawiera sześć pól: licznik powtórzeń, kod OEM, flagę rozszerzonej klawiatury, kod kontekstu, poprzedni stan klawisza i stan przejściowy.
Licznik powtórzeń zawiera liczbę naciśnięć klawiszy; dla komunikatów związanych z zwalnianiem klawisza zawsze równa się jeden.
(pobieramy licznik powtórzeń – LOWORD(lParam))

wParam – kod klawisza wirtualnego.

Copyright © 2001-2003 Ewa Lach

Dziesiętnie	*Szest.*	*Identyfikator w windows.h*	*Klawiatura IBM*
3	03	VK_CANCEL	Ctrl +Break
8	08	VK_BACK	Backspace
9	09	VK_TAB	Tab
12	0C	VK_CLEAR	Numeric keyboard 5 with Num Lock OFF
13	0D	VK_RETURN	Enter (either one)
16	10	VK_SHIFT	Shift (either one)
17	11	VK_CONTROL	Ctrl (either one)
18	12	VK_MENU	Alt (either one)
19	13	VK_PAUSE	Pause
20	14	VK_CAPITAL	Caps Lock
27	1B	VK_ESCAPE	Esc
32	20	VK_SPACE	Spacebar
33	21	VK_PRIOR	Page Up
34	22	VK_NEXT	Page Down
35	23	VK_END	End
36	24	VK_HOME	Home
37	25	VK_LEFT	Left Arrow
38	26	VK_UP	Up Arrow
39	27	VK_RIGHT	Right Arrow
40	28	VK_DOWN	Down Arrow
41	29	VK_SELECT	
42	2A	VK_PRINT	
43	2B	VK_EXECUTE	
44	2C	VK_SNAPSHOT	Print Screen
45	2D	VK_INSERT	Insert
46	2E	VK_DELETE	Delete
47	2F	VK_HELP	
48-57	30-39	None	0 - 9 z głównej części klawiatury
65_90	41-5A	None	A - Z
96-105	60-69	VK_NUMPAD0-VK_NUMPAD9	Klawiatura numeryczna 0-9 z włączonym Num Lock
106	6A	VK_MULTIPLY	Klawiatura numeryczna: [*]
107	6B	VK_ADD	Klawiatura numeryczna: [+]
108	6C	VK_SEPARATOR	
109	6D	VK_SUBTRACT	Klawiatura numeryczna:[-]
110	6E	VK_DECIMAL	Klawiatura numeryczna: [.]
111	6F	VK_DIVIDE	Klawiatura numeryczna: [/]
112-121	70-79	VK_F1 - VK_F10	Klawisz funkcyjny F1 - F10
122-135	7A-87	VK_F11 - VK_F24	Klawisz funkcyjny F11 - F24
144	90	VK_NUMLOCK	Num Lock
145	91	VK_SCROLL	Scroll Lock

Copyright © 2001-2003 Ewa Lach

Aby sprawdzić w jakim stanie były klawisze specjalne w trakcie naciśnięcia dowolnego klawisza można skorzystać z funkcji:
`SHORT` **`GetKeyState`**`(int nVirtKey)`
która zwraca wartość ujemną jeżeli był naciśnięty klawisz [Shift], [Ctrl] lub [Alt], oraz zwraca wartość 0 jeżeli nie są włączone: [Caps Lock], [Num Lock] oraz [Scroll Lock].

Komunikaty znakowe
Aby stwierdzić jaki znak odpowiada naciśniętemu klawiszowi często potrzebna jest znajomość językowej konfiguracji klawiatury. Dlatego należy wykorzystać Windows do tłumaczenia komunikatów klawiszy na komunikaty znakowe. Robi to funkcja *`TranslateMessage`* wywoływana w pętli komunikatów. *`TranslateMessage`* umieszcza komunikat znakowy w kolejce komunikatów. Mamy cztery komunikaty znakowe odpowiadające czterem komunikatom klawiszy.
W większości przypadków program zajmuje się tylko jednym: `WM_CHAR`.
`WM_CHAR` :
`wParam` – kod znaku. (Znaki specjalne ‘`\b`’ – [Backspace], ‘`\t`’ – [Tab],
‘`\n`’ – nowy wiersz, ‘`\r`’- powrót karetki)
`lParam` – jak w `WM_KEYDOWN`.

Aby wyświetlić więcej niż jeden wiersz potrzebujemy informacji na temat wielkości znaków czcionki używanej w oknie.
Czcionka jest określana dla danego kontekstu urządzenia. Domyślna czcionka nosi nazwę `SYSTEM_FONT` (czcionka systemowa). Od Windows 3.0 czcionka systemowa ma zmienną szerokość znaków.
Wymiarów znaku dostarcza funkcja `GetTextMetrics`.
`BOOL` **`GetTextMetrics`**`(HDC hdc, LPTEXTMETRIC lptm)`
`hdc` –uchwyt kontekstu urządzenia
`lptm` – wskaźnik na strukturę TEXTMETRIC, którą funkcja wypełnia informacjami o czcionce wybranej w kontekście urządzenia.

Struktura **`TEXTMETRIC`** określa wielkości charakteryzujące czcionkę. Jednostki, w których są one wyrażone zależą od odwzorowania wybranego w kontekście urządzenia. Domyślne odwzorowanie to `MM_TEXT`, a więc wszystkie wymiary są podane w pikselach. Użyteczne pola tej struktury to:
`LONG tmHeight` – wysokość znaku
`LONG tmAveCharWidth` – średnia ważona szerokość małych liter (najczęściej definiowana jako szerokość x)
`LONG tmMaxCharWidth` – szerokość najszerszego znaku w czcionce

Ilość tekstu, który się mieści w obszarze roboczym zależy od wielkości obszaru roboczego. Możemy określić wymiar obszaru roboczego okna przejmując komunikat `WM_SIZE`.
`WM_SIZE` - wysyłany po każdej zmianie wielkości okna. Jeżeli w klasie okna zdefiniowano style `CS_HREDRAW | CS_VREDRAW` to po tym komunikacie pojawia się komunikat `WM_PAINT`.
`HIWORD(lParam)` – wysokość obszaru roboczego
`LOWORD(lParam)` – szerokość obszaru roboczego
lub korzystając z funkcji, która zwraca współrzędne obszaru roboczego okna w stosunku do lewego górnego rogu obszaru roboczego okna (`lpRect.left = lpRect.top = 0`)
`BOOL` **`GetClientRect(`**`HWND hWnd, LPRECT lpRect)`

Copyright © 2001-2003 Ewa Lach

Obsługa komunikatów myszy

WM_MOUSEMOVE - komunikat umieszczany w kolejce gdy kursor myszy zmienił położenie w obszarze roboczym okna, lub jeśli mysz została przechwycona (*SetCapture*) bez względu na jej położenie.
`wParam` - informacja o naciśniętych klawiszach wirtualnych
kombinacja wartości:
`MK_CONTROL, MK_LBUTTON, MK_MBUTTON, MK_RBUTTON, MK_SHIFT`
`LOWORD(lParam)` - pozioma pozycja kursora względem lewego górnego rogu obszaru roboczego
`HIWORD(lParam)` - pionowa pozycja kursora

WM_LBUTTONDOWN/ WM_RBUTTONDOWN - komunikat umieszczany w kolejce gdy naciśnięto lewy / prawy klawisz myszy w obszarze roboczym okna, interpretacja parametrów jak w komunikacie `WM_MOUSEMOVE`.

WM_LBUTTONUP/ WM_RBUTTONUP - komunikat umieszczany w kolejce gdy zwolniono lewy / prawy klawisz myszy w obszarze roboczym okna, interpretacja parametrów jak w komunikacie WM_MOUSEMOVE.

WM_LBUTTONDBLCLK/ WM_RBUTTONDBLCLK- komunikat umieszczany w kolejce gdy naciśnięto dwukrotnie lewy/ prawy klawisz myszy w obszarze roboczym okna, interpretacja parametrów jak w komunikacie `WM_MOUSEMOVE`. Aby okno otrzymywało ten komunikat styl okna musi mieć wartość `CS_DBLCLKS`.

Aby sprawdzić w trakcie obsługiwania komunikatów myszy czy wciśnięty jest dany przycisk myszy, [Shift] lub [Ctrl] możemy wykonać koniunkcję bitową:
```
if(MK_SHIFT & wParam )
     [klawisz [Shift] jest wciśnięty ]
```

Aby obejrzeć obsługę komunikatów myszy skorzystamy z funkcji GDI:
`BOOL` **`MoveToEx`**`(HDC hdc, int X, int Y, LPPOINT lpPoint)` - ustawia bieżącą pozycję pióra
`lpPoint` – zawiera poprzednią pozycję. Jeżeli nie interesuje nas ta wartość możemy podać NULL.
Możemy pobrać bieżącą pozycję za pomocą:
`BOOL` **`GetCurrentPositionEx`**`(HDC hdc, LPPOINT lpPoint);`

`BOOL` **`LineTo`**`(HDC hdc, int nXend, int nYend)` – rysuje linie od bieżącej pozycji do punktu podanego w funkcji (ale bez tego punktu). Zmienia bieżącą pozycję pióra na punkt podany w funkcji.

Czasami mamy do czynienia z sytuacją gdy nie chcemy aby okno straciło kontakt z myszą. Kiedy użytkownik ciągnie wciśniętą mysz, chwilowe wysunięcie wskaźnika poza okno nie powinno powodować żadnych kłopotów. Program powinien stale kontrolować ruch myszy. Aby przechwycić mysz wystarczy wywołać funkcję **`SetCapture(hwnd)`**. Po tym wywołaniu Windows wysyła wszystkie komunikaty myszy do procedury tego okna, którego uchwyt równa się hwnd. Mysz zwalniamy jednym wywołaniem **`ReleaseCapture().`**

Copyright © 2001-2003 Ewa Lach

Aby ochronić się przed programami, które przechwytują mysz, ale jej nie zwalniają wprowadzono ograniczenie, że jeżeli mysz została przechwycona, ale jej przycisk nie jest wciśnięty, a przesuniemy wskaźnik myszy do innego okna, to komunikaty myszy zacznie otrzymywać okno pod wskaźnikiem.

Komunikat Zegara.

Zegar jest urządzeniem wejścia, które cyklicznie informuje aplikację o upływie określonego interwału czasu. Program decyduje o długości tego interwału.

Zegar zanim będzie można z niego skorzystać, musi zostać przydzielony do programu. Zadanie to spełnia funkcja *SetTimer*.

```
UINT SetTimer(HWND hWnd, UINT nIDEvent, UINT uElapse,
              TIMERPROC lpTimerFunc)
```

`hWnd` - uchwyt okna, które ma otrzymywać komunikaty. Okno musi należeć do danej aplikacji. Jeżeli ten parametr jest równy `NULL`, to żadne okno nie jest związane z zegarem, a parametr `nIDEvent` jest ignorowany.
`nIDEvent` - identyfikator zegara (!=0). Jeżeli `hWnd =NULL`, parametr jest ignorowany. Jeżeli `hWnd!=0` i okno o uchwycie `hWnd` już posiada zegar o wartości `nIDEvent` to istniejący zegar jest zamieniany przez nowo utworzony.
`uElapse` - długość interwału w milisekundach (1000 – sekunda)

`lpTimerFunc` – adres funkcji wywoływanej zwrotnie (call-back function) do obsługi komunikatu `WM_TIMER`. Funkcja musi być zdefiniowana jako `CALLBACK`; jej parametry wejściowe są takie same jak procedury okna; funkcja nie zwraca żadnej wartości. Np.:

```
void CALLBACK TimerProc(HWND hwnd, UINT iMsg, UINT iTimerId,
                        DWORD dwTime)
```

`iMsg` – identyfikator komunikatu
`iTiemerId` – identyfikator zegara
`dwTime` – ilość milisekund które upłynęły od rozpoczęcia działania systemu
Wywołanie *SetTimer* ma wtedy postać:

```
SetTimer(hWnd, nIDEvent, uElapse, (TIMERPROC)TimerProc)
```

Jeżeli ten parametr równa się NULL to Windows wysyła `WM_TIMER` do procedury okna.

Jeżeli funkcja zakończy się pomyślnie i `hWnd=NULL` to funkcja zwróci identyfikator zegara typu integer, Jeżeli `hWnd!=NULL` to funkcja zwróci wartość różną od zera. Jeśli nie ma w systemie wolnego zegara funkcja zwraca `NULL`.

Gdy zegar przestaje być potrzebny program powinien wywołać funkcję `KillTimer`, która wstrzymuje nadsyłanie komunikatów zegara. Funkcja ta powoduje także usunięcie z kolejki komunikatów `WM_TIMER`.

```
BOOL KillTimer(HWND hWnd, UINT uIDEvent)
```

`hWnd` - uchwyt okna - właściciela zegara. Jeżeli w `SetTimer hWnd =NULL` to tutaj też powinien równać się `NULL`.
`uIDEvent` - identyfikator zegara.

Copyright © 2001-2003 Ewa Lach

WM_TIMER - komunikat jest umieszczany w kolejce jeśli zainstalowano zegar funkcją *SetTimer* i upłynął zadany kwant czasu. Komunikat podobnie jak WM_PAINT ma niski priorytet i zostaje wysłany do okna dopiero gdy kolejka jest pusta. W danej chwili w kolejce może znajdować się tylko jeden komunikat WM_TIMER (tak jak WM_PAINT).
wParam - identyfikator zegara
lParam - wskaźnik na procedurę obsługi zegara

funkcja zwracająca lokalny czas i datę:
```
VOID GetLocalTime(LPSYSTEMTIME lpSystemTime)
```

funkcja, która zamienia napis na pasku tytułowym okna:
```
BOOL SetWindowText(HWND hWnd, LPCTSTR lpString);
```

Komunikaty pasków przewijania.

Konieczność użycia pasku przewijania pojawia się w momencie gdy brakuje miejsca w oknie. Aby umieścić pionowy lub poziomy pasek przewijania w oknie aplikacji należy wstawić w stylu okna w funkcji *CreateWindow* – WS_VSCROLL (przewijanie pionowe), WS_HSCROLL (przewijanie poziome).

Z każdym paskiem przewijania wiąże się „zakres" – para liczb całkowitych wyznaczających wartość minimalną i maksymalną. Domyślnie zakres wyznaczają liczby 0 oraz 100. Można zmienić zakres za pomocą funkcji *SetScrollRange*.
```
BOOL SetScrollRange(HWND hWnd, int nBar, int nMinPos,
                    int nMaxPos, BOOL bRedraw)
```
hwnd - uchwyt okna, którego pasek przewijania ma być zmieniony lub uchwyt paska przewijania (dla pasków tworzonych funkcją CreateWindow) w zależności od parametru nBar.
nBar - określa który pasek okna będzie modyfikowany (SB_HORZ lub SB_VERT), dla samodzielnych pasków należy podać wartość SB_CTL (wtedy hwnd jest uchwytem paska przewijania)
nMinPos - nowa minimalna pozycja suwaka
nMaxPos - nowa maksymalna pozycja suwaka
fRedraw - parametr określający czy należy odświeżyć pasek na ekranie

Z każdym paskiem przewijania wiąże się także pozycja suwaka, określana zawsze jako liczba całkowita. Funkcja *SetScrollPos* umożliwia zmianę położenia suwaka.
```
int SetScrollPos(HWND hWnd, int nBar, int nPos, BOOL bRedraw)
```
hWnd, nBar, bRedraw - tak samo jak w *SetScrollRange*.
nPos - nowa pozycja suwaka

Za pomocą funkcji *GetScrollRange* i *GetScrollPos* możesz pobrać aktualny zakres i pozycję suwaka.
```
int GetScrollPos(HWND hWnd, int nBar);
BOOL SetScrollRange(HWND hWnd, int nBar, int nMinPos,
                    int nMaxPos);
```

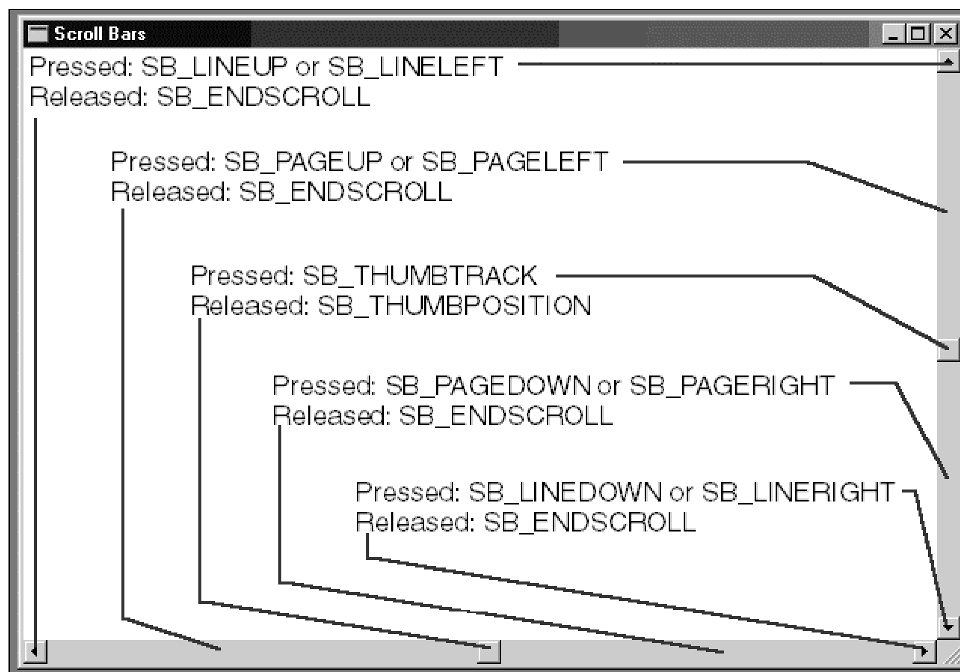
Copyright © 2001-2003 Ewa Lach

`WM_HSCROLL, WM_VSCROLL` - komunikaty od pasków przewijania

`lParam - = 0` dla pasków przewijania okna (dla okien w stylu z flagą `WS_HSCROLL` lub `WS_VSCROLL`), lub uchwyt okna sterowania dla pozostałych.

`(int) LOWORD(wParam)` - kod operacji :

```
#define SB_LINEUP           0
#define SB_LINELEFT         0
#define SB_LINEDOWN         1
#define SB_LINERIGHT        1
#define SB_PAGEUP           2
#define SB_PAGELEFT         2
#define SB_PAGEDOWN         3
#define SB_PAGERIGHT        3
#define SB_THUMBPOSITION    4
#define SB_THUMBTRACK       5
#define SB_TOP              6
#define SB_LEFT             6
#define SB_BOTTOM           7
#define SB_RIGHT            7
#define SB_ENDSCROLL        8
```



Jeżeli chcemy natychmiast zaktualizować unieważniony obszar bezpośrednio po *`InvalidateRect`* należy wywołać *`UpdateWindow`*, która powoduje natychmiastowe wywołanie procedury okna z komunikatem `WM_PAINT`. W tym przypadku `WM_PAINT` omija kolejkę komunikatów.

Copyright © 2001-2003 Ewa Lach

Do przewijania i odświeżania obszaru roboczego możemy wykorzystać funkcję *ScrollWindowEx*. Funkcja unieważnia tylko prostokątny wycinek obszaru roboczego, który w wyniku przewinięcia został odsłonięty.

```
int ScrollWindowEx ( HWND hWnd, int dx, int dy,
                     CONST RECT *prcScroll,
                     CONST RECT *prcClip, HRGN hrgnUpdate,
                     LPRECT prcUpdate, UINT flags)
```

hWnd – uchwyt okna, którego obszar roboczy chcemy przewinąć

dx – wielkość przesunięcia w poziomie w jednostkach urządzenia. W przypadku przesunięcia w lewo wartość jest ujemna

dy – wielkość przesunięcia w pionie w jednostkach urządzenia. W przypadku przesunięcia do góry wartość jest ujemna

prcScroll – wskaźnik na prostokąt obszaru roboczego który ma być skrolowany. Jeżeli =NULL cały obszar roboczy ulega przesunięciu.

prcClip – wskaźnik na prostokąt, który ma podlegać zmianie w wyniku skrolowania. Część przesunięta z zewnątrz prostokąta do wewnątrz jest rysowana, natomiast część przesuwana z wnętrza prostokąta na zewnątrz nie jest rysowana. Ten parametr może przyjmować wartość NULL.

hrgnUpdate – uchwyt do regionu który podlega modyfikacji

prcUpdate – wskaźnik na prostokąt który zostaje wypełniony przez funkcję informacjami o unieważnionym obszarze.

flags – ustawia flagi które kontrolują przewijanie. Może przyjmować następujące wartości:

SW_INVALIDATE – unieważnia region określony przy pomocy parametru hrgnUpdate po przesunięciu.

SW_ERASE – czyści unieważniony region, jeżeli został wyspecyfikowany.

SW_SCROLLCHILDREN – przesuwa okna potomne znajdujące się w obszarze określonym prcScroll

Copyright © 2001-2003 Ewa Lach

Funkcje i komunikaty do zarządzania oknami

WM_GETMINMAXINFO - komunikat generowany kiedy rozmiar lub pozycja okna mają ulec zmianie
lParam - wskaźnik na strukturę **MINMAXINFO** opisującą okno

```
typedef struct tagMINMAXINFO {
POINT ptReserved;
POINT ptMaxSize;          // maksymalna szerokość (point.x) i wysokość okna
                          // w postaci zmaksymalizowanej
POINT ptMaxPosition;      // pozycja lewego górnego rogu okna w postaci
                          // zmaksymalizowanej
POINT ptMinTrackSize;     // minimalne rozmiary okna
POINT ptMaxTrackSize;     // maksymalne rozmiary okna
} MINMAXINFO;
```

WM_WINDOWPOSCHANGED - komunikat jest wysyłany do okna którego rozmiar, pozycja lub położenie Z uległo zmianie (na skutek wywołania funkcji SetWindowPos lub innej funkcji zarządzającej oknami)
lParam - wskaźnik na strukturę **WINDOWPOS** opisującą pozycje i rozmiar okna.

```
typedef struct _WINDOWPOS {
HWND hwnd;                // uchwyt okna
HWND hwndInsertAfter;     // pozycja na osi Z ( od przodu do tyłu –
                          // oś głębokość).może to być uchwyt do okna
                          // znajdującego się przed oknem do którego wysłano
                          // komunikat, lub jedna z wartości parametru
                          // hWndInsertAfter funkcji SetWindowPos.
int x;                    // pozycja lewej krawędzi okna
int y;                    // pozycja górnej krawędzi okna
int cx;                   // szerokość okna
int cy;                   // wysokość okna
UINT flags;               // dodatkowe flagi określające stan okna
} WINDOWPOS;
```

WM_SIZE – komunikat wysyłany do okna którego rozmiar uległ zmianie
wParam – może przyjąć wartości:
SIZE_MAXIMIZED – okno zostało zmaksymalizowane
SIZE_MINIMIZED – okno zostało zminimalizowane
SIZE_RESTORED – rozmiar okna uległ zmianie (nie doszło ani do maksymalizacji ani do minimalizacji)
LOWORD(lParam) – szerokość obszaru roboczego okna
HIWORD(lParam) – wysokość obszaru roboczego okna

WM_MOVE – komunikat wysyłany do okna którego położenie uległo zmianie.
LOWORD(lParam) – pozycja lewej krawędzi okna
HIWORD(lParam) – pozycja górnej krawędzi okna

Copyright © 2001-2003 Ewa Lach

Funkcja zmieniająca rozmiar, pozycję lub położenie Z okna.

```
BOOL SetWindowPos(HWND hWnd, HWND hWndInsertAfter,int X,int Y,
                  int cx, int cy, UINT nFlags)
```

hWnd – uchwyt okna

hwndInsertAfter – uchwyt okna które znajduje się na oknie które zmieniamy lub jedna z wartości:

- HWND_BOTTOM – umieszcza okno za wszystkimi innymi oknami
- HWND_NOTOPMOST – umieszcza okno za oknami z opcją zawsze na wierzchu, ale przed oknami bez tej opcji.
- HWND_TOP – umieszcza okno na wierzchu
- HWND_TOPMOST - umieszcza okno przed oknami z opcją zawsze na wierzchu i sam otrzymuje tę opcję.

X – pozycja lewej krawędzi okna

Y – pozycja górnej krawędzi okna

cx – szerokość okna

cy – długość okna

nFlags – może kombinacją następujących wartości:

- SWP_DRAWFRAME – rysuje ramkę zdefiniowaną w opisie klasy okna dookoła okna
- SWP_FRAMECHANGED – ustawia nowy styl ramki, zmienionej przy pomocy funkcji *SetWindowLong*
- SWP_HIDEWINDOW – ukrywa okno. Jeżeli ta flaga jest ustawiona okno nie może zmienić pozycji lub rozmiaru.
- SWP_NOACTIVATE – nie czyni okna aktywnym. W przeciwnym wypadku okno jest aktywowane i przesunięte na wierzch.
- SWP_NOMOVE – nie zmienia pozycji okna. Ignoruje parametry X i Y.
- SWP_NOOWNERZORDER – nie zmienia pozycji właściciela okna na osi Z.
- SWP_NOREDRAW – nie odświeża po zmianie. Odnosi się do całego okna (obszar roboczy + pasek tytułowy + paski przewijania+..) oraz do części okna rodzica odkrytej po zmianie położenia okna.
- SWP_NOREPOSITION = SWP_NOOWNERZORDER
- SWP_NOSENDCHANGING – okno nie otrzymuje komunikatu WM_WINDOWPOSCHANGING
- SWP_NOSIZE – nie zmienia rozmiaru okna. Ignoruje parametry cx i cy.
- SWP_NOZORDER – nie zmienia pozycji na osi Z.
- SWP_SHOWWINDOW – wyświetla okno. Jeżeli ta flaga jest ustawiona okno nie może zmienić pozycji lub rozmiaru.

Funkcja zmieniająca rozmiar i pozycję okna:

```
BOOL MoveWindow(HWND hWnd, int X, int Y, int cx, int cy, BOOL
bRepaint)
```

hWnd, X, Y, cy, cy – jak w funkcji *SetWindowPos*

bRepaint – Jeżeli równa się false nie następuje odświeżenie po zmianie. Odnosi się do całego okna (obszar roboczy + pasek tytułowy + paski przewijania+..) oraz do części okna rodzica odkrytej po zmianie położenia okna.

Copyright © 2001-2003 Ewa Lach

Funkcja, która rozmieszcza wyspecyfikowane okna potomne określonego rodzica.

```
WORD TileWindows(HWND hwndParent, UINT wHow,
                 CONST RECT *lpRect, UINT cKids,
                 const HWND *lpKids)
```

`hwndParent` – uchwyt okna rodzica, jeżeli równy NULL za okno rodzica uznaje się desktop.

`wHow` – Przyjmuje jedną z wartości:

- `MDITILE_HORIZONTAL` – rozmieszcza okna w poziomie
- `MDITILE_VERTICAL` – rozmieszcza okna w pionie

`LpRect` – wskaźnik na prostokątny obszar w ramach którego rozmieszcza się okna. Jeżeli równy NULL cały obszar roboczy okna jest wykorzystywany.

`cKids` – liczba elementów tablicy określonej w parametrze `lpKids`. Parametr jest ignorowany jeżeli `lpKids=NULL`

`lpKids` – tablica uchwytów okien potomnych. Jeżeli parametr równa się NULL wszystkie okna potomne są rozmieszczane.

Funkcja , która minimalizuje okno

```
BOOL CloseWindow( HWND hWnd);
```

Funkcja która przywraca poprzednią wielkość i pozycję okna

```
BOOL OpenIcon(HWND hWnd)
```

Funkcja dostarcza informacji na temat określonego okna:

```
BOOL GetWindowInfo( HWND hwnd, PWINDOWINFO pwi)
```

`hwnd`- uchwyt okna o którym chcemy pobrać informacje.

`pwi` – wskaźnik na strukturę **WINDOWINFO**, którą funkcja wypełnia informacjami na temat okna.

```
typedef struct tagWINDOWINFO {
DWORD cbSize,             \\ rozmiar struktury
RECT rcWindow,            \\ wskaźnik na prostokąt, który zawiera współrzędne okna
RECT rcClient,            \\ wskaźnik na prostokąt, który zawiera współrzędne
                          \\ obszaru roboczego
DWORD dwStyle,            \\ style okna
DWORD dwExStyle,          \\ rozszerzone style okna
DWORD dwWindowStatus,     \\ przyjmuje wartość WS_ACTIVECAPTION jeżeli okno
                          \\ jest aktywne, w przeciwnym razie 0
UINT cxWindowBorders,     \\ wielkość ramki okna w pikselach
UINT cyWindowBorders,     \\ wysokość ramki okna w pikselach
ATOM atomWindowType,      \\ atom klasy okna (RegisterClass)
WORD wCreatorVersion,     \\ wersja Windowsów pod którymi została stworzona
                          \\ aplikacja, która utworzyła to okno.
} WINDOWINFO;
```

Funkcja która zwraca współrzędne okna, w stosunku do lewego, górnego rogu ekranu.

```
BOOL GetWindowRect(HWND hWnd, LPRECT lpRect);
```

Copyright © 2001-2003 Ewa Lach

Funkcja, która zwraca współrzędne obszaru roboczego okna w stosunku do lewego górnego rogu obszaru roboczego okna `(lpRect.left = lpRect.top = 0)`
`BOOL` **`GetClientRect(`**`HWND hWnd, LPRECT lpRect)`

Funkcja zwraca uchwyt desktopu.
`HWND` **`GetDesktopWindow`**`(VOID);`

Funkcja zwraca uchwyt okna, które jest w specjalnej relacji z oknem podanym jako parametr funkcji.
`HWND` **`GetWindow`**`(HWND hWnd, UINT uCmd);`
uCmd - Jeżeli równy GW_OWNER to funkcja zwraca właściciela okna.

`HWND` **`GetParent`**`(HWND hWnd)` – funkcja zwraca uchwyt rodzica okna identyfikowanego za pomocą *hWnd*, jeżeli jest to okno typu CHILD, w przeciwnym wypadku zwraca uchwyt właściciela okna lub jeżeli okno nie ma właściciela NULL.

Funkcja, która zamienia napis na pasku tytułowym okna.
`BOOL` **`SetWindowText`**`(HWND hWnd, LPCTSTR lpString);`

Copyright © 2001-2003 Ewa Lach