Okna sterujące

Okna sterujące – kontrolki – to okna potomne, które otrzymują i obsługują wszystkie komunikaty myszy i klawiatury oraz informują okno nadrzędne o zdarzeniach związanych ze zmianą swojego stanu. Okno potomne jest urządzeniem wejścia dla okna nadrzędnego. Można utworzyć własne okna potomne, ale warto korzystać z predefiniowanych klas okien standardowych kontrolek.

Pracę z kontrolkami zaczynamy od ich utworzenia przy pomocy funkcji ***CreateWindow***. Nie musimy rejestrować klas tych okien, klasy te już istnieją w Windows. Są to klasy o nazwach: `button` (przycisk), „**static**" (pole tekstowe), "**scrollbar**" (pasek przewijania), „**edit**" (pole edycji), „**listbox**" (pole listy), „**combobox**" (lista rozwijana z możliwością edycji). Następnie za pomocą funkcji ***MoveWindow*** umieszczamy kontrolki w wybranym miejscu w oknie nadrzędnym. Ostatni etap to przechwytywanie komunikatu `WM_COMMAND`, który kontrolka wysyła do okna nadrzędnego.

`BUTTON` (przycisk)

Tworząc przycisk przy pomocy funkcji `CreateWindow` oprócz przypisania pierwszemu parametrowi lpClassName wartości „button" musimy określić styl przycisku (w parametrze `dwStyle`).

Przycisk może mieć następujące style (`BS_`):

`BS_PUSHBUTTON` - przycisk

`BS_DEFPUSHBUTTON` – wskazuje na przycisk domyślny (odznacza się czarniejszą ramką)

`BS_CHECKBOX` - pole wyboru

`BS_AUTOCHECKBOX` - pole wyboru z automatyczną aktualizacją stanu

`BS_RADIOBUTTON` - pole selekcji

`BS_3STATE` – trzy- stanowe pole wyboru (zaznaczony, odznaczony, zaznaczony – przydymiony)

`BS_AUTO3STATE` – trzy - stanowe pole wyboru z automatyczną aktualizacją stanu

`BS_GROUPBOX` - ramka grupująca inne okna sterujące. Nie wysyła, ani nie odbiera żadnych komunikatów.

`BS_AUTORADIOBUTTON` - pole selekcji z automatyczną aktualizacją stanu

`BS_OWNERDRAW` –przycisk, którego wyświetlanie spoczywa na programie. Dopóki nie określimy jego wyglądu przycisk zaznacza swoją obecność w obszarze okna nadrzędnego szarym prostokątem Przycisk informuje o potrzebie malowania go wysyłając komunikat `WM_DRAWITEM` z parametrem lParam wskazującym na strukturę typu DRAWITEMSTRUCT

`BS_LEFTTEXT` – styl występujący w kombinacji z stylem radio-button lub check-box. Powoduje, że tekst ukazuje się po lewej stronie przycisku, zamiast po prawej (jak jest domyślnie).

Jeżeli nie określimy żadnego z powyższych styli domyślnie Windows przyjmuje styl: `BS_PUSHBUTTON`.

Aby przycisk się ukazał oprócz powyższych styli musi zostać określony styl : `WS_CHILD | WS_VISIBLE`.

Po kliknięciu przycisku okno potomne wysyła do rodzica komunikat **`WM_COMMAND`**.

`LOWORD(wParam)` – ID (identyfikator) okna potomnego – określony w *`CreateWndow (hMenu)`*

`HIWORD(wParam)` – Kod powiadomienia. W większości programów kod powiadomienia jest równy: `BN_CLICKED` z wartością 0.

`lParam` – uchwyt okna potomnego.

Po kliknięciu przycisku, wokół umieszczonego na nim napisu pojawia się ramka wykreślona linią przerywaną. Oznacza to, że przycisk ma ognisko. Wszystkie polecenia z klawiatury wędrują teraz do przycisku, który ignoruje wszystkie naciśnięcia klawiszy z wyjątkiem spacji, której naciśnięcie daje taki sam efekt jak kliknięcie.
W momencie utraty ogniska procedura okna nadrzędnego otrzymuje komunikat `WM_KILLFOCUS`. Możemy powstrzymać okno przed utratą ogniska wywołując w procedurze obsługi tego komunikatu funkcję:
`HWND` **`SetFocus`**`(HWND hwnd)` –funkcja daje oknu o uchwycie *hwnd* ognisko i zwraca uchwyt okna, które utraciło ognisko.

Procedura okna może wysyłać komunikatu do okna potomnego kontrolki. Wymaga to znajomości uchwytu okna potomnego.
Funkcje zwracające ID okna:
`GetWindowLong``(hwndChild, GWL_ID);`
`GetDlgCtrlID``(hwndChild);`
Funkcje zwracające uchwyt okna potomnego:
`GetDlgItem``(hwndParent, id);`

Komunikaty wysyłane przez okno nadrzędne do przycisku(BM_):

- `BM_SETSTATE` – wysyłając ten komunikat można programowo symulować naciśnięcie przycisku:
  `SendMessage(hWndButtton, BM_SETSTATE,1,0);`

i jego zwolnienie:
  `SendMessage(hWndButtton, BM_SETSTATE,0,0);`

- `BM_GETSTATE` – funkcja SendMessage zwróci TRUE – jeśli przycisk jest wciśnięty, FALSE – w przeciwnym wypadku
- `BM_GETCHECK` – informuje czy pole wyboru / przycisk radiowy jest ustawiony czy nie.
- `BM_SETCHECK` – ustawia pole wyboru / przycisk radiowy jeżeli trzeci parametr jest równy 1; jeżeli trzeci parametr jest równy 0 odznacza przycisk.
- `BM_SETSTYL` – pozwala na zmianę stylu przycisku po jego utworzeniu.

`wParam` – określa nowy styl przycisku
`LOWORD(lParam)` – TRUE – powoduje odmalowanie przycisku, FLASE – przycisk nie jest rysowany ponownie. Można skorzystać z makra
`LPARAM` **`MAKELPARAM`**`(WORD wLow, WORD wHigh)`

Etykietę przycisku możemy zmienić przy pomocy funkcji okna:
`SetWindowText``(hwnd, pszString);`
Do pobrania etykiety służy funkcja:
`GetWindowText``(hwnd, pszBuffer, GetWindowLength(hwnd));`

Aby okno potomne otrzymywało informacje wejściowe musi być widoczne i dostępne.

Jeśli podczas tworzenia okna nie uwzględnimy stylu WS_VISIBLE w klasie okna, okno potomne będzie niewidoczne dopóki nie wywołasz funkcji:
`BOOL ShowWindow(HWND hwnd, int mCmdShow);`
mCmdShow wskazuje w jaki sposób okno ma być pokazane. Nas interesują wartości:
`SW_SHOWNORMAL` – pokazuje okno
`SW_HIDE` – ukrywa okno
`BOOL IsWindowVisible(HWND hwnd)` – sprawdza czy okno jest wyświetlone

Można udostępniać i blokować dostęp do okna za pomocą funkcji:
`BOOL EnableWindow(HWND hwnd, BOOL bEnable);`
`bEnable` – określa czy udostępnić (TRUE), czy zablokować dostęp do okna (FALSE)
Można sprawdzić dostępność do okna za pomocą funkcji:
`BOOL IsWindowEnabled(HWND hwnd)`

W przypadku przycisków `OWNERDRAW` program sam zajmuje się ich wyświetlaniem. Komunikat **`WM_DRAWITEM`** informuje program, że przycisk potrzebuje wykreślenia. Dzieje się to w chwili: utworzenia przycisku, jego wciśnięcia lub zwolnienia, w momencie otrzymania lub utraty ogniska i kiedy potrzebne jest odświeżenie ekranu.
`lParm` jest wskaźnikiem na strukturę **`DRAWITEMSTRUCT`**, która dostarcza informacji niezbędnych do wykreślenia kontrolki.

```
typedef struct tagDRAWITEMSTRUCT {
UINT CtlType,  // określa typ kontrolki (ODT_BUTTON, ODT_COMBOBOX,
ODT_LISTBOX,…)
UINT CtlID,    // określa identyfikator okna potomnego
UINT itemID,   // używany przy menu, liście rozwijanej z edycją oraz polu listy.
UINT itemAction,      // określa jaki rodzaj odrysowania jest konieczny
                // ODA_DRAWENTIRE – cała kontrolka musi być odświeżona,
                // ODA_FOCUS – kontrolka otrzymała lub straciła ognisko
                // ODA_SELECT – zmienił się stan kontrolki.
UINT itemState,       // określa stan kontrolki (patrz niżej)
HWND hwndItem, // uchwyt okna sterowania
HDC hDC,       // uchwyt kontekstu urządzenia kontrolki
RECT rcItem    // wymiary przycisku
ULONG_PTR itmeData  // używany przy menu, liście rozwijanej z edycją oraz polu listy
} DRAWITEMSTRUCT;
```

`itemState` może być kombinacją następujących wartości: *(sprawdzamy itemState & ODS_...)*

```
ODS_CHECKED         - Pozycja menu ma być zaznaczona.
ODS_COMBOBOXEDIT- Rysowanie ma miejsce w polu edycji listy rozwijanej.
ODS_DEFAULT         - Kontrolka jest kontrolką domyślną.
ODS_DISABLED        - Kontrolka ma być rysowana z zablokowanym dostępem
ODS_FOCUS           - Kontrolka ma ognisko
ODS_GRAYED          - Pozycja menu ma być narysowana na szaro.
ODS_SELECTED        - Kontrolka została zaznaczona.
```

Korzystając z uchwytu kontekstu urządzenia kontrolki możemy narysować kontrolkę. W większości aplikacji umieszcza się na ich powierzchni małe mapy bitowe.
My narysujemy kontrolkę korzystając z kilku funkcji GDI.
`int` **`FillRect`**`(HDC hDC, CONST RECT *lprc, HBRUSH hbr)`
– funkcja wypełnia prostokąt lprc przy pomocy pędzla zdefiniowanego za pomocą jego uchwytu hbr.
Uchwyt do pędzla otrzymujemy za pomocą funkcji **GetStockObject,** która zwraca uchwyt predefiniowanych pędzli, palet, czcionek. Jako parametr przyjmuje ona wartość która specyfikuje co ma być zwrócone. Nas interesuje: BLACK_BRUSH, WHITE_BRUSH, GRAY_BRUSH, LTGRAY_BRUSH (light)
`int` **`FrameRect(`**`HDC hDC, CONST RECT *lprc, HBRUSH hbr)` - funkcja rysuje ramkę dookoła podanego prostokąta.
`int` **`InvertRect(`**`HDC hDC, CONST RECT *lprc)` - funkcja odwraca kolory w zadanym prostokącie.
`BOOL` **`DrawFocusRect(`**`HDC hDC, CONST RECT *lprc)` - funkcja kreśli kropkowaną ramkę.
Stosując uchwyt kontekstu z DRAWITEMSTRUT należy pamiętać o zwalnianiu obiektów wybranych w tym kontekście i nie rysowaniu poza obszarem przycisku.

STATIC (statyczne okna potomne)
Style SS_:
SS_BLACKRECT - prostokąt wypełniony kolorem czarnym
SS_BLACKFRAME - ramka czarna
SS_GRAYRECT - prostokąt wypełniony kolorem szarym
SS_GRAYFRAME - ramka szara
SS_WHITERECT - prostokąt wypełniony kolorem białym
SS_WHITEFRAME - ramka biała
SS_ETCHEDHORZ, SS_ETCHEDVER, SS_ETCHEDFRAME - ramka z „cieniem”
SS_LEFT, SS_RIGHT, SS_CENTER – wyrównanie napisu statycznego (można zmieniać funkcją SetWindowText)

`WM_CTLCOLORSTATIC` – komunikat wysyłany przez okno statyczne oraz pole edycji z zablokowanym dostępem lub typu read-only, informujący o potrzebie wykreślenia kontrolki. Obsługując ten komunikat program może zmienić tło oraz kolor tekstu statycznego okna
`wParam` – uchwyt kontekstu urządzenia statycznego okna
`lParam` – uchwyt statycznego okna.
Obsłużony komunikat zwraca pędzel dla tła okna.
`COLORREF` **`SetBkColor`**`(HDC hdc, COLORREF crColor)` – funkcja zmienia kolor tła tekstu i zwraca poprzedni kolor tła.
Aby otrzymać obiekt typu COLORREF można użyć makra:
`COLORREF` **`RGB`**`(BYTE byRed, BYTE byGreen, BYTE byBlue);`
`COLORREF` **`SetTextColor`**`(HDC hdc, COLORREF crColor)` – zmienia kolor czcionki

SCROLLBAR (pasek przewijania)
Style SBS_
SBS_VERT - pasek przewijania pionowy
SBS_HORZ - pasek przewijania poziomy
W przeciwieństwie do pozostałych kontrolek paski przewijania wysyłają komunikaty WM_VSCROLL i WM_HSCROLL.
Większość funkcji używanych do obsługi pasków przewijania jest taka sama jak dla pasków przewijania okien.

WM_CTLCOLORSCROLLBAR – komunikat wysyłany rodzicowi paska przewijania (kontrolki) kiedy kontrolka potrzebuje wykreślenia. Obsługując ten komunikat rodzic może zmienić tło paska przewijania. Obsłużony komunikat zwraca pędzel dla tła.

Jeżeli chcemy obsłużyć niektóre komunikaty przychodzące do procedury okna kontrolek, możemy tego dokonać zmieniając adres procedury okna dla danej kontrolki. (technika „zakładania podklasy okna")

`DWORD` **`SetWindowLong`**`(HWND hWnd, int nIndex, LONG dwNewLong)` - funkcja umożliwia zmianę niektórych atrybutów danego okna. Zwraca wartość sprzed zamiany.

hWnd – uchwyt okna

nIndex – określa wartość do zamiany.

Aby zamienić adres procedury okna należy użyć: GWL_WNDPROC

dwNewLong – określa nową wartość.

Nowa procedur okna powinna wywoływać poprzednią procedurę okna, która obsłuży pozostałe komunikaty:

```
return CallWindowProc(stara_proc,hwnd, iMsg, wParam,lParam);
```

stara_proc – wartość typu WNDPROC zwrócona przez *SetWindowLong*.

EDIT (pole edycji)
Style ES_:
ES_LEFT, ES_RIGHT, ES_CENTER - wyrównywanie tekstu
ES_MULTILINE – kontrolka wielowierszowa (domyślne jednowierszowa)
ES_AUTOHSCROLL - przewijanie tekstu w kierunku poziomym, razem ze stylem MULTILINE przejście do następnego wiersza jedynie po naciśnięciu [ENTER]. Aby dodać pasek przewijania: WS_HSCROLL.
ES_AUTOVSCROLL – w kontrolkach wielowierszowych umożliwia przewijanie tekstu w pionie. Aby dodać pasek przewijania: WS_VSCROLL.
ES_NOHIDESEL - nie usuwa zaznaczenia (wizualnie) po utracie ogniska
Standardowo kontrolka edycji nie ma krawędzi. Ich dołączenie wymaga użycia stylu WS_BORDER.

Kontrolka wysyła do okna nadrzędnego komunikat **WM_COMMAND**.
lParam – uchwyt okna potomnego
LOWORD(wParam) – identyfikator okna potomnego.
HIWORD(wParam) – Kod powiadomienia:
EN_SETFOCUS – kontrolka otrzymała ognisko wejścia
EN_KILLFOCUS – kontrolka utraciła ognisko wejścia
EN_CHANGE – zawartość pola edycji uległa zmianie
EN_UPDATE - zawartość pola edycji ulegnie zmianie
EN_ERRSPACE – Kontrolka nie ma więcej miejsca (pojemność kontrolki ograniczona do 32 KB)
EN_MATEXT – Brak miejsca na wstawienie więcej znaków (ograniczona do 30,000 znaków tekstu)
EN_HSCROLL - Kliknięto poziomy pasek przewijania kontrolki
EN_VSCROLL - Kliknięto pionowy pasek przewijania kontrolki

Aby wstawić tekst do pola edycji używamy *SetWindowText*, aby go pobrać – *GetWindowText* razem z *GetWindowTextLength*.

Komunikaty wysyłane do pola edycji

Zaznaczanie fragmentu tekstu:
```
SendMessage(hwndEdit, EM_SETSEL, iStart ,iEnd);
```
Usuwanie zaznaczonego tekstu i wstawianie go do schowka:
```
SendMessage(hwndEdit, WM_CUT, 0, 0);
```
Kopiowanie zaznaczonego tekstu do schowka:
```
SendMessage(hwndEdit, WM_COPY, 0, 0);
```
Usuwanie zaznaczonego tekstu:
```
SendMessage(hwndEdit, WM_CLEAR, 0, 0);
```
Wstawianie tekstu z schowka w miejsce wskazane kursorem:
```
SendMessage(hwndEdit, WM_PASTE, 0, 0);
```
Zastąpienie zaznaczonego teksu dowolnym innym tekstem:
```
SendMessage(hwndEdit, WM_REPLACESEL, 0, (LPARAM) szString);
```
Ustalenie początku i końca zaznaczonego tekstu:
```
SendMessage(hwndEdit, EM_GETSEL, (WPARM) &iStart ,
                    (LPARAM) &iEnd);
```
Sprawdzenie liczby wierszy z wielowierszowej kontrolki:
```
iCount = SendMessage(hwndEdit, EM_GETLINECOUNT, 0, 0);
```

Sprawdzenie przesunięcia wiersza (liczby znaków poprzedzających pierwszy znak w wierszu)

```
iOffset = SendMessage(hwndEdit, EM_LINEINDEX, iLine, 0);
```

Jeżeli `iLine=-1` otrzymujemy przesunięcie wiersza dla linii, w której znajduje się kursor.

Sprawdzenie długości wiersza:

```
iLength = SendMessage(hwndEdit, EM_LINELENGTH, iLine, 0);
```

Wstawienie kopii wiersza do bufora:

```
SendMessage(hwndEdit, EM_GETLINE, iLine, (LPARAM) szBurffer);
```

LISTBOX (pole listy)
Style LBS_:
LBS_NOTIFY - okno nadrzędne ma otrzymywać komunikat WM_COMMAND. Domyślnie nie otrzymuje tego komunikatu.
LBS_SORT - alfabetyczny porządek listy.
LBS_MULTIPLESEL - możliwość wyboru wielu elementów listy. Domyślnie można wybrać tylko jeden element.
LBS_NOREDRAW – nie aktualizuje ekran po każdej zmianie listy. (zmieniamy opcję komunikatem: WM_SETREDRAW).
LBS_STANDARD – jest to kombinacja styli : LBS_NOTIFY | LBS_SORT | WS_BORDER | WS_VSCROLL
LBS_OWNERDRAWFIXED – program jest odpowiedzialny za rysowanie tej kontrolki. Pozycje na liście mają taką samą wysokość.
LBS_OWNERDRAWVARIABLE – program jest odpowiedzialny za rysowanie tej kontrolki. Pozycje na liście mają różną wysokość.

Do tekstów związanych z poszczególnymi polami listy odwołujemy się przez indeks (indeks pierwszej pozycji na liście równa się 0)

Komunikaty wysyłane do pola listy:

Dodawanie elementu do listy posortowanej

```
SendMessage(hwndList, LB_ADDSTRING, 0, (LPARM) szString);
```

Dodawanie elementu do listy nieposortowanej:

```
SendMessage(hwndList, LB_INSERTSTRING, iIndex, (LPARM)
szString);
```

Jeżeli iIndex=-1 tekst zostanie dopisany na końcu listy

Usuwanie pozycji listy:

```
SendMessage(hwndList, LB_DELETESTRING, iIndex, 0);
```

Czyszczenie całej listy

```
SendMessage(hwndList, LB_RESETCONTENT, 0, 0);
```

Wyłączenie/włączanie opcji aktualizacji ekranu po każdej zmianie:

```
SendMessage(hwndList, WM_SETREDRAW, FALSE/TRUE , 0);
```

Sprawdzenie liczebności listy:

```
iCount=SendMessage(hwndList, LB_GETCOUNT, 0, 0);
```

Wypełnienie pola listy zawartością bieżącego lub podanego we wzorcu katalogu:

```
SendMessage(hwndList, LB_DIR, iAttr, (LPARAM)szFileSpec);
```

iAttr- określa co ma być dodane do listy

- DDL_READWRITE – Zwykły plik (plik normalny, plik tylko do odczytu, pliki do archiwzacji)
- DDL_READONLY – Plik tylko do odczytu
- DDL_HIDEN – Plik ukryty
- DDL_SYSTEM – plik systemowy
- DDL_DIRECTORY – podkatalog i katalog nadrzędny :[..]
- DDL_ARCHIVE – plik z ustawionym bitem archiwizacji
- DDL_DRIVES – wyświetla litery dysków
- DDL_EXCLUSIVE – przeszukiwanie wykluczające (pliki z oznaczonymi flagami, ale z wykluczeniem zwykłych plików)

lParam – wskaźnik do wzorca pliku np.: „*.*”, „c:*.*”

funkcja zmieniająca bieżący katalog (z biblioteki direct.h) : `int chdir(const char * dirname)`

Lista z możliwością zaznaczania tylko jednego wiersza:

Wyróżnienie pozycji na liście:

```
SendMessage(hwndList, LB_SETCURSEL, iIndex, 0);
```

Jeżeli `iIndex=-1` żadna z pozycji nie będzie wyróżniona.

Wybranie pozycji na podstawie jej znaków początkowych:

```
SendMessage(hwndList, LB_SELECTSTRING, iIndex,
(LPARAM)szSearchString);
```

iIndex – indeks pozycji od której Windows rozpoczyna wyszukiwanie. Jeżeli `=-1` to szuka od początku listy.

Funkcja zwraca indeks wybranej pozycji lub jeżeli nie znajdzie pozycji – LB_ERR.

Sprawdzenie indeksu aktualnie wybranej pozycji listy:

```
iIndex=SendMessage(hwndList, LB_GETCURSEL, 0, 0);
```

Sprawdzenie ilości znaków tekstu określanej pozycji listy:

```
iLength=SendMessage(hwndList, LB_GETTEXTLEN, iIndex, 0);
```

Wstawienie tekstu (o zwracanej długości) do bufora:

```
iLength=SendMessage(hwndList, LB_GETTEXT, iIndex,
(LPARAM)szBuffer);
```

Lista z możliwością zaznaczania więcej niż jednego wiersza:

Zaznaczanie pozycji na liście niezależnie od stanu pozostałych pozycji:

```
SendMessage(hwndList, LB_SETSEL, wParam, iIndex);
```

wParam – wyznacza stan pozycji. `!=0` – pozycja zostanie wybrana.

Jeżeli `iIndex=-1` wszystkie pozycje na liście zostaną zaznaczone jako wybrane lub nie wybrane.

Sprawdzenie stanu pozycji

```
iSelect=SendMessage(hwndList, LB_GETSEL, iIndex, 0);
```

iSelect !=0 – pozycja wybrana

Okno nadrzędne otrzymuje od pola listy komunikat : WM_COMMAND o znaczeniu parametrów j.w.

Kod powiadomienia	Wartość	Opis
LBN_ERRSPACE	-2	w kontrolce pola listy nie ma więcej miejsca
LBN_SELCHANGE	1	nastąpiła zmiana wybranej pozycji
LBN_DBLCLK	2	nastąpiło dwukrotne kliknięcie myszą na pozycję listy
LBN_CANCEL	3	użytkownik cofnął zaznaczenie pozycji listy
LBN_SETFOCUS	4	lista otrzymała ognisko
LBN_KILLFOUCS	5	lista utraciła ognisko

SendMessage zwraca: jeżeli wystąpi błąd spowodowany wyczerpaniem pamięci na pola listy - LB_ERRSPACE(-2), inny rodzaj błędu - LB_ERR(-1), operacja zakończona powodzeniem – LB_OKAY.

COMBOBOX (lista z polem edycji)
style CBS_:
CBS_SORT - alfabetyczny porządek listy
CBS_UPPERCASE - tylko duże litery
CBS_SIMPLE - lista jest rozwinięta cały czas
CBS_AUTOHSCROLL – automatycznie przewija tekst w polu edycji kontrolki w prawo. Jeżeli ten styl nie jest ustawiony, tylko tekst, który mieści się w polu edycji jest dozwolony.
CBS_DROPDOWN – lista jest rozwijana, kiedy użytkownik naciśnie na przycisk obok pola edycji.
CBS_DROPDOWNLIST – j.w. tylko pole edycji jest zastąpione oknem statycznym, które wyświetla bieżący wybór w oknie.
CBS_NOINTEGRALHEIGHT – wskazuje, że rozmiar kontrolki jest dokładnie taki jak podany w programie przy tworzeniu listy. Domyślnie, Windows zmienia rozmiar kontrolki tak aby lista nie wyświetlała niecałych pozycji.
CBS_OWNERDRAWFIXED – program jest odpowiedzialny za rysowanie tej kontrolki. Pozycje na liście mają taką samą wysokość.
CBS_OWNERDRAWVARIABLE – program jest odpowiedzialny za rysowanie tej kontrolki. Pozycje na liście mają różną wysokość.

Komunikaty wysyłane do pola listy:

Dodawanie elementu do listy posortowanej

```
SendMessage(hwndList, CB_ADDSTRING, 0, (LPARM) szString);
```

Dodawanie elementu do listy nieposortowanej:

```
SendMessage(hwndList, CB_INSERTSTRING, iIndex, (LPARM)
szString);
```

Jeżeli `iIndex=-1` tekst zostanie dopisany na końcu listy

Usuwanie pozycji listy:

```
SendMessage(hwndList, CB_DELETESTRING, iIndex, 0);
```

Czyszczenie całej listy i pola edycji

```
SendMessage(hwndList, CB_RESETCONTENT, 0, 0);
```

Sprawdzenie liczebności listy:

```
iCount=SendMessage(hwndList, CB_GETCOUNT, 0, 0);
```

Wypełnienie pola listy zawartością bieżącego katalogu:

```
SendMessage(hwndList, CB_DIR, iAttr, (LPARAM)szFileSpec);
```
Parametry jak dla listbox.

Wyróżnienie pozycji na liście i skopiowanie jej tekstu do pola edycji:
```
SendMessage(hwndList, CB_SETCURSEL, iIndex, 0);
```
Jeżeli `iIndex=-1` żadna z pozycji nie będzie wyróżniona.

Wybranie pozycji na podstawie jej znaków początkowych i wpisanie do pola edycji:
```
SendMessage(hwndList, CB_SELECTSTRING, iIndex,
(LPARAM)szSearchString);
```
iIndex – indeks pozycji od której Windows rozpoczyna wyszukiwanie. Jeżeli `=-1` to szuka od początku listy.
Funkcja zwraca indeks wybranej pozycji lub jeżeli nie znajdzie pozycji – LB_ERR.

Sprawdzenie indeksu aktualnie wybranej pozycji listy:
```
iIndex=SendMessage(hwndList, CB_GETCURSEL, 0, 0);
```
Sprawdzenie ilości znaków tekstu określanej pozycji listy:
```
iLength=SendMessage(hwndList, CB_GETLBTEXTLEN, iIndex, 0);
```
Wstawienie tekstu (o zwracanej długości) do bufora:
```
iLength=SendMessage(hwndList, CB_GETLBTEXT, iIndex,
(LPARAM)szBuffer);
```

WM_COMMAND

Kod powiadomienia	Opis
CBN_ERRSPACE	- w kontrolce nie ma więcej miejsca
CBN_SELCHANGE	- nastąpiła zmiana wybranej pozycji na liscie
CBN_DBLCLK	- nastąpiło dwukrotne klikniecie myszą na pozycję listy
CBN_SETFOCUS	- lista otrzymała ognisko
CBN_KILLFOUCS	- lista utraciła ognisko
CBN_EDITCHANGE	- zawartość pola edycji uległa zmianie
CBN_EDITUPDATE	- zawartość pola edycji ulegnie zmianie

Okna dialogowe

Okna dialogowe używa się najczęściej do pobrania dodatkowych informacji od użytkownika.
Okna dialogowe dzielimy na:
modalne – kiedy program wyświetla takie okno, użytkownik nie może przełączyć się do innego okna w programie.
niemodalne – użytkownik może przełączyć się do innego okna w programie

Okno dialogowe jest zasobem i definiujemy je w skrypcie zasobów.

```
nameID DIALOG x, y, cy, cy
[[optional-statements]]
{ control-statement . . . }
```

Np.:

```
AboutBox DIALOG 32, 32, 180, 102
STYLE DS_MODALFRAME | WS_POPUP
FONT 8, "MS Sans Serif"
{
   DEFPUSHBUTTON            "OK",    IDOK, 66, 81,    50,  14,
WS_GROUP
   ICON       "ABOUT1",      IDC_STATIC, 7, 7,  21,    20
   CTEXT      "About1",      IDC_STATIC, 40,    12,    100, 8
   CTEXT      "Przykładowe okno",     IDC_STATIC, 7,   40,
   166,       8
}
```

Pierwszy wiersz nadaje oknu dialogowemu nazwę („AboutBox” – można też użyć liczby) oraz wymiary (x,y,cx,cy).
x – współrzędna lewego górnego rogu okna dlg. na osi x, liczona względem obszaru roboczego okna nadrzędnego, wyrażona w jednostkach odpowiadających `1/4` szerokości znaku czcionki systemowej.
y – współrzędna lewego górnego rogu okna dlg. na osi y, liczona względem obszaru roboczego okna nadrzędnego, wyrażona w jednostkach odpowiadających `1/8` wysokości znaku czcionki systemowej.
cx – szerokość okna dialogowego wyrażona w jednostkach odpowiadających `1/4` szerokości znaku czcionki systemowej.
cy – wysokość okna dialogowego wyrażona w jednostkach odpowiadających `1/8` wysokości znaku czcionki systemowej.

Wymiary czcionki systemowej można otrzymać za pomocą funkcji:

```
iCxCy=GetDialogBaseUnits();
```

LOWORD(iCxCy) – szerokość znaku czcionki systemowej
HIWORD(iCxCy) – wysokość znaku czcionki systemowej

Po słowie kluczowym **STYLE** określamy styl okna dialogowego, korzystając ze styli okna WS_. Jeżeli nasza deklaracja nie zwiera linii STYLE domyślnie przyjmowany jest styl: WS_POPUP | WS_DLGFRAME.

Możemy określić także tytuł okna (jeżeli podaliśmy styl WS_CAPTION) po słowie kluczowym CAPTION. Np.:

```
CAPTION „Tytuł okna dialogowego”
```

Po słowie kluczowym **FONT** określamy rozmiar i nazwę czcionki (innej niż systemowa) dla tekstów pojawiających się w oknie dialogowym.

Do okna dialogowego możemy także dodać menu, pisząc:

```
MENU nazwaMenu
```

Możemy także określić klasę okna dialogowego, która m.in. wskazuje systemowi, że komunikaty ma wysyłać do procedury okna określonej w klasie.

```
CLASS „nazwaKlasy”
```

Wewnątrz nawiasów klamrowych definiujemy okna potomne kontrolek. Format jest następujący

```
typ_kontrolki  „tekst”, id, xPos, yPos, xWidth, yHeight [, iStyle]
```

Typ kontrolki	**Klasa**	**Styl Windows**
(Skrótowy zapis dla okien potomnych kontrolek)		
PUSHBUTTON	button	BS_PUSHBUTTON ¦ WS_TABSTOP
DEFPUSHBUTTON	button	BS_DEFPUSHBUTTON ¦ WS_TABSTOP
CHECKBOX	button	BS_CHECKBOX ¦ WS_TABSTOP
RADIOBUTTON	button	BS_RADIOBUTTON ¦ WS_TABSTOP
GROUPBOX	button	BS_GROUPBOX ¦ WS_TABSTOP
LTEXT	static	SS_LEFT ¦ WS_GROUP - ramka grupy.
CTEXT	static	SS_CENTER ¦ WS_GROUP
RTEXT	static	SS_RIGHT ¦ WS_GROUP
ICON	static	SS_ICON
EDITTEXT	edit	ES_LEFT ¦ WS_BORDER ¦ WS_TABSTOP
SCROLLBAR	scrollbar	SBS_HORZ
LISTBOX	listbox	LBS_NOTIFY ¦ WS_BORDER ¦ WS_VSCROLL
COMBOBOX	combobox	CBS_SIMPLE ¦ WS_TABSTOP

Wszystkie wyżej wymienione kontrolki dodatkowo mają styl: WS_CHILD | WS_VISIBLE

„text” zależy od typu kontrolki, dla pola tekstu, przycisku jest to tekst pojawiający się na nim, dla ikony jest to *nazwaID* ze skryptu zasobów definiująca ikonę. Dla pola edycji, list i pasków przewijania ten parametr jest pomijany (drugi parametr jest id).

id – identyfikator okna potomnego, wykorzystywany przy wysyłaniu komunikatów do okna nadrzędnego. Jeżeli kontrolka nie wysyła, ani nie otrzymuje komunikatu ustawiamy tę wartość na `-1`.

Definicja xPos, yPos , xWidth i yHeight jest taka sama jak x,y,cx,cy. W przypadku ikony wielkości te są ignorowane.

Można także użyć ogólnej instrukcji definiowania kontrolek:

```
CONTROL   „tekst”, id, ”nazwa_klasy”, iStyl, xPos, yPos, xWidth, yHeight
```

W programie musimy umieścić procedurę okna dialogowego, która obsługuje komunikaty skierowane do tego okna. Nie jest to jednak prawdziwa procedura okna. Procedura okna dla okna dialogowego znajduje się w Windows. Wywołuje ona naszą procedurę okna dialogowego.

```
BOOL CALLBACK DlgProc(HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM wParam);
```

Zwraca TRUE jeśli przetwarza komunikat , jeśli nie – zwraca FALSE.

WM_INITDIALOG – komunikaty wysyłany zaraz przed wyświetleniem okna dialogowego bezpośrednio do procedury okna dialogowego. Program wykorzystuje obsługę tego komunikatu najczęściej do inicjalizacji, które mają wpływ na wygląd dialogu.
LOWORD(wParam) – identyfikator kontrolki domyślnej (pierwszej kontrolki w dialogu, która jest widoczna, baz zablokowanego dostępu i ma ustawiony styl WS_TABSTOP) która ma otrzymać ognisko jeżeli funkcja zwróci TRUE.
Jeżeli nie chcemy, aby ognisko otrzymała kontrolka domyślna funkcja powinna zwrócić FALSE.

Funkcja wywołująca modalne okno dialogowe:

```
int DialogBox(HINSTANCE hInstance, LPCTSTR lpTemplate, HWND
hWndParent,DLGPROC lpDialogFunc);
```

hInstance - uchwyt wystąpienia
lpTemplate - nazwa okna dialogowego, zdefiniowana w skrypcie
hWndParent - uchwyt okna właściciela dialogu
lpDialogFunc - wskaźnik na procedurę obsługi dialogu

Funkcja zamykająca okno dialogowe, otwarte za pomocą funkcji *DialogBox*:

```
BOOL EndDialog(HWND hDlg, int nResult);
```

hDlg - uchwyt okna dialogowego
nResult - tą wartość zwróci funkcja *DialogBox*

Jeżeli mamy grupę przycisków radiowych z identyfikatorami tworzącymi ciąg rosnący możemy zaznaczyć jeden przycisk i odznaczyć pozostałe wywołując funkcję (zamiast wysyłać do wszystkich BM_SETCHECK):

```
BOOL CheckRadioButton(HWND hwnd, int nIDFirstButton, int
nIDLastButton, int nIDCheckButton);
```

Jeżeli w dialogu utworzymy przycisk z predefiniowanym identyfikatorem równym **IDOK** (także jeżeli użyjemy wartość 1) to program będzie interpretował naciśnięcie klawisza [Enter], jeżeli żaden inny przycisk ([def]pushbutton) nie ma ogniska wejścia, tak jakby naciśnięto na przycisk o tym identyfikatorze. W przypadku **IDCANCEL** (2) – niezależnie od tego kto posiada ognisko klawisz [Esc] będzie interpretowany jako naciśnięcie klawisza o id=2.

W przypadku okna dialogowego Windows bierze na siebie obsługę przenoszenia ogniska z jednej kontrolki do drugiej, po naciśnięciu odpowiednich klawiszy.
Kontrolka z stylem **WS_GROUP** rozpoczyna nową grupę, która obejmuje wszystkie kontrolki, aż do kolejnego wywołania WS_GROUP.
Kontrolka ze stylem **WS_TABSTOP** może otrzymać ognisko po naciśnięciu klawisza [Tab]
Niektóre kontrolki dialogu mają domyślnie ustawiony styl WS_TABSTOP i WS_GROUP.

Windows umożliwia poruszanie się między grupami za pomocą klawisza [Tab] i w ramach grupy za pomocą przycisków nawigacyjnych.
Znak **&** w tekście powoduje, że możemy przekazywać ognisko wybranym kontrolkom naciskając na literę następującą po &.

Możemy zdefiniować własną kontrolkę, tworząc własną klasę okna, rejestrując ją i następnie podając jej nazwę w linii CONTROL w opisie szablonu dialogu w skrypcie zasobów. W podanej w klasie procedurze okna przechwytujemy WM_PAINT, aby narysować kontrolkę, oraz przesyłamy oknu nadrzędnemu o zdarzeniach, które chcemy aby zostały obsłużone. W przypadku komunikatów nie obsłużonych procedura okna zwraca funkcję *DefWindowProc*

Niemodalne okna dialogowe

Okno niemodalne wywołujemy za pomocą funkcji
```
HWND CreateDialog(HINSTANCE hInstance, LPCTSTR lpTemplate,
HWND hWndParent, DLGPROC lpDialogFunc);
```
Parametry funkcji są takie same jak w DialogBox.
Różnice:
funkcja zwraca uchwyt do okna niemodalnego.
Przy definiowaniu stylu okna niemodalnego powinniśmy dodać WS_CAPTION (można przesunąć okno), WS_SYSMENU (można korzystać z tego menu np. do zamknięcia tego okna), WS_VISIBLE (bez tego okno nie zostanie wyświetlone: chyba, że wywołamy później funkcje ShowWindow(hDlgModeless, SW_SHOW)
Ponieważ komunikaty okna modalnego przechodzą przez kolejkę komunikatów trzeba dodać w pętli:
```
if(hDlgModless == 0 || !IsDialogMessage(hDlgModless, &msg))
    [wyślij komunikaty do procedury okna]
```
hDlgModless – zmienna globalna, której przypisujemy uchwyt otwartego okna niemodalnego
IsDialogMessage – funkcja wysyłająca komunikat do procedury okna dialogowego i zwracająca TRUE jeżeli komunikat jest przeznaczony dla niemodalnego okna dialogowego, w przeciwnym wypadku zwraca FALSE.
Do zamknięcia niemodalnego okna używamy funkcji:
DestroyWindow(HWND hDlg);
Okno rodzica powinno być stylu : WS_CLIPCHILDREN aby nie rysować po oknie niemodalnym.

Przydatne funkcje:
Funkcja przypisuje polu tekstowemu kontrolki okna dialogowego liczbę
```
BOOL SetDlgItemInt(HWND hDlg, int nIDDlgItem, UINT uVlue, BOOL
bSigned);
```
hDlg – uchwyt okna dialogowego
nIDDlgItem – identyfikator kontrolki
uVlue – wartość
bSigned – określa czy uValue jest signed czy unsigned int.

Funkcja przekształca tekst określonej kontrolki na wartość liczbową:
```
UINT GetDlgItemInt(HWND hDlg, int nIDDlgItem, BOOL
*lpTranslated, BOOL bSigned);
```
lpTranslated – wskaźnik na obiekt który może przyjmuje wartość True jeżeli funkcja zakończyła działanie sukcesem.

Główne okno aplikacji – okno dialogowe
W szablonie okna dialogowego w skrypcie zasobów wprowadzamy linię: CLASS „nazwa_klasy_okna";
W głównej klasie okna parametrowi cbWndExtra przypisujemy wartość : DLGWINDOWEXTRA;
Zamiast CreateWindow wywołujemy
```
CreateDialog(hInstance,lpTemplate,0,NULL);
```

Standardowe okna dialogowe:
COMDLG.H – plik nagłówkowy zawierający definicję funkcji, które wywołują standardowe okna dialogowe służące do otwierania i zapisywania plików, wyszukiwania i zastępowania tekstu, wyboru koloru i czcionek, a także drukowania.

Aby zainicjalizować standardowe dialogi otwierania i zapisywania pliku należy zadeklarować zmienną wskazującą na strukturę typu **OPENFILENAME**. Struktura przechowuje informacje, które umożliwiają inicjalizację dialogu. Następnie system zwraca informacje na temat wyboru użytkownika w tej strukturze.

```
typedef struct tagOFN {
DWORD lStructSize      // = sizeof (OPENFILENAME)
HWND hwndOwner         // uchwyt okna właściciela lub NULL
HINSTANCE hInstance    // = NULL
LPCTSTR lpstrFilter    // wskaźnik na bufor zawierający pary zakończonych zerem
                       // wzorców plików i ich nazw. Ciąg  znaków musi być zakończony
                       // podwójnym zerem. Pierwszy w parze opisuje filtr .(np. Plik
                       // tekstowy) drugi – specyfikuje wzorzec( np. *.txt)-nie może
                       // zawierać spacji, zgodnie z którym zostaną wyświetlone pliki w
                       // oknie dialogowym.
                       // "Text\0*.txt\0Rys.\0*.jpg;*.bmp;*.gif\0All
                       //  F.(*.*)\0*.*\0\0"
LPTSTR lpstrCustomFilter   // = NULL
DWORD nMaxCustFilter       // = 0 ;
DWORD nFilterIndex     // wskazuje, który wzorzec (filtr) ma być wyświetlony ( liczy od 1)
LPTSTR lpstrFile       // wskaźnik do bufora w celu otrzymania pełnej nazwy ścieżki. Przy
                       //inicjacji może = NULL
DWORD nMaxFile         // rozmiar bufora lpstrFile
LPSTR lpstrFileTitle   // wskaźnik do bufora w celu otrzymania samej nazwy pliku.
                       // Przy inicjacji może = NULL
DWORD nMaxFileTitle    // rozmiar bufora lpstrFileTitle
LPSTR lpstrInitialDir  // wskaźnik na bufor specyfikujący początkowy katalog.
                       // W zależności od systemu sposób  wyboru początkowego katalogu
                       // jest różny
LPCTSTR lpstrTitle     // wskaźnik na bufor zawierający tytuł dialogu. Jeżeli = NULL
                       // Windows użyje domyślny
```

```
DWORD Flags          // Flagi, które użytkownik może użyć do inicjalizacji dialogu.
                     // Po zamknięciu dialogu  wskazują one na akcje użytkownika. Patrz
                     //niżej.
WORD nFileOffset     // = 0
WORD nFileExtension  // =0
LPCTSTR lpstrDefExt  // wskaźnik na bufor który zawiera domyślne rozszerzenie nazwy
                     // pliku, jeśli użytkownik  nie poda własnego wpisując nazwę pliku
                     // do okna dialogowego (np. „exe”)
LPARAM lCustData     // = 0L ;
LPOFNHOOKPROC lpfnHook       // = NULL ;
LPCTSTR lpTemplateName // = NULL ;
}*LPOPENFILENAME;
```

Flags (wybrane):
OFN_ALLOWMULTISELECT – można wybrać więcej niż jeden plik; lpstrFile zwraca ścieżkę do bieżącego katalogu, po której następują nazwy wybranych plików.
OFN_CREATEPROMPT – Jeżeli użytkownik podaje plik, który nie istnieje, ta flaga powoduje, że dialog pyta użytkownika o pozwolenie utworzenia tego pliku. Jeżeli użytkownik decyduje się utworzyć plik dialog ulega zamknięciu, a funkcja zwraca podaną nazwę w przeciwnym wypadku dialog pozostaje otwarty.
OFN_EXTENSIONDIFFERENT – wskazuje, że użytkownik wpisał inne rozszerzenie pliku niż jest podane w lpstrDefExt.
OFN_FILEMUSTEXIST – wskazuje, że użytkownik może podać nazwę jedynie istniejącego pliku.
OFN_HIDEREADONLY – Domyślne okno dialogowe File Open zawiera pole wyboru, które pozwala użytkownikowi zdecydować, czy plik ma zostać otwarty tylko do czytania. Flaga ta powoduje że nie wyświetlane jest to pole.
OFN_NOCHANGEDIR – przywraca bieżący katalog po tym jak użytkownik go zmienił szukając pliku.
OFN_OVERWRITEPROMPT – powoduje, że dialog Save As pyta się użytkownika o pozwolenie zastąpienia istniejącego pliku.
Funkcja wyświetlająca standardowy dialog otwierania pliku (File Open)

```
BOOL GetOpenFileName(LPOPENFILENAME lpofn);
```

Funkcja zwraca wartość różną od 0 jeżeli użytkownik wybrał plik i nacisnął przycisk OK

Funkcja wyświetlająca standardowy dialog zapisywania pliku (File Save)

```
BOOL GetSaveFileName(LPOPENFILENAME lpofn);
```

Funkcja zwraca wartość różną od 0 jeżeli użytkownik wybrał plik i nacisnął przycisk OK

CHOOSEFFONT - Struktura przechowuje informacje, które umożliwiają inicjalizację dialogu czcionek przy pomocy funkcji *ChooseFont*. Następnie system zwraca informacje na temat wyboru użytkownika w tej strukturze.

```
typedef struct {
DWORD lStructSize    //= sizeof (CHOOSEFONT)
HWND hwndOwner    // uchwyt okna właściciela lub NULL
HDC Hdc              // uchwyt kontekstu urządzenia drukarki. Ignorujemy.
LPLOGFONT lpLogFont     // wskaźnik na strukturę LOGFONT. Jeżeli użytkownik naciśnie
                     //OK. w dialogu czcionek funkcja ChooseFont wypełnia tę strukturę
                     // inf. o wybranej przez użytkownika czcionce.
INT iPointSize       // wielkość zaznaczonej czcionki  ustawiana przez ChooseFont
DWORD Flags          // Flagi, które użytkownik może użyć do inicjalizacji dialogu. Po
                     // zamknięciu dialogu  wskazują one na akcje użytkownika. Patrz
                     // niżej.
COLORREF rgbColors       // Określa kolor czcionki
LPARAM lCustData    // = 0L ;
LPCFHOOKPROC lpfnHook         // = NULL ;
LPCTSTR LpTemplateName        // nazwa szablonu dialogu = NULL ;
HINSTANCE hInstance      // uchwyt wystąpienia dla pliku zawierającego szablon dialogu =
                         //NULL ;
LPTSTR lpszStyle      // wskaźnik na bufor, który zawiera informacje na temat stylu = NULL
WORD nFontType        //Zwrócone przez ChooseFont, określa rodzaj czcionki:
                      // BOLD_FONTTYPE , ITALIC_FONTTYPE,
                      //  REGULAR_FONTTYPE (normalna),
                      // PRINTER_FONTTYPE (czcionka drukarki),
                      // SCREEN_FONTTYPE,
                      // SIMULATED_FONTTYPE ( czcionka jest imitowana przez GDI)
INT nSizeMin          // określa minimalną wielkość czcionki użytkownik może wybrać = 0
INT nSizeMax          //określa maksymalną wielkość czcionki użytkownik może wybrać = 0
}*LPCHOOSEFONT;
```

Flags (wybrane):
CF_TTONLY – wskazuje, że funkcja ChooseFont pozwala na wybór jedynie wśród czcionek TrueType.
CF_EFFECTS – powoduje, że dialog wyboru czcionki wyświetla kontrolki, które umożliwiają wybranie użytkownikowi opcji podkreślenia, przekreślenia oraz koloru czcionki.

```
typedef struct tagLOGFONT {
  LONG lfHight       // wysokość czcionki. Jeżeli =0 system używa domyślnej wielkości
  LONG lfWidth       // szerokość
  LONG lfEscapement
  LONG lfOrientation
  LONG lfWeight      // grubość czcionki (np. FW_DONTCARE=0, FW_NORMAL=400,
                     //FW_BOLD=700,..)
  BYTE lfItalic
  BYTE lfUnderline
  BYTE lfStrikeOut
  BYTE lfCharSet
  BYTE lfOutPrecision
  BYTE lfClipPrecision
```

```
  BYTE  lfQuality
  BYTE lfPitchAndFamily
  TCHAR lfFaceName
} LOGFONT, *PLOGFONT
```

funkcja wywołująca dialog wyboru czcionki:
`BOOL` **`ChooseFont`**`(LPCHOOSEFONT lpcf);`

Okna komunikatów – alternatywa dla okien dialogowych, jeżeli potrzebujemy prostej odpowiedzi użytkownika.
`int` **`MessageBox`**`(HWND hWnd, LPCTSTR lpText, LPCTSTR lpCaption, UINT uType)` – funkcja wyświetla okno komunikatów.
hWnd - uchwyt okna właściciela lub NULL
lpText - wskaźnik na tekst komunikatu
lpCaption - wskaźnik na tekst nagłówka okna,
uType - flagi stylu okna komunikatu

- flagi które określają przyciski, które pojawią się na dole okna:
  - MB_OK. (domyślny),
  - MB_OKCANCEL,
  - MB_YESNO,
  - MB_YESNOCANCEL,
  - MB_RETRYCANCEL,
  - MB_ABORTRETRYIGNORE,
  - MB_HELP
- flagi, które określają, który przycisk jest domyślny:
  - MB_DEFBUTTON1 (domyślny), MB_DEFBUTTON2, MB_DEFBUTTON3, MB_DEFBUTTON4
- flagi, które określają jaka ikona pojawi się w oknie komunikatu:
  - MB_ICON = MB_ICONINFORMATION,
  - MB_ICONEXCLAMATION = MB_ICONWARNING,
  - MB_ICONERROR = MB_ICONSTOP = MB_ICONHAND,
  - MB-ICONQUESTION
- flagi określające modalność okna:
  - MB_APLMODAL – domyślna – modalna ze względu na aplikację – użytkownik może przejść do innej aplikacji bez zamykania okna komunikatu.
  - MB_SYSTEMMODAL – modalna systemowo - użytkownik nie może przejść do innej aplikacji bez zamykania okna komunikatu.
- flaga, która powoduje że okno komunikatu nie otrzymuje ogniska:
  - MB_NOFOCUS.

funkcja zwraca jedną z wartości odpowiadającą naciśnietemu przyciskowi:
IDABORT, IDCANCEL, IDIGNORE ,IDNO, IDOK, IDRETRY, IDYES, IDHELP.