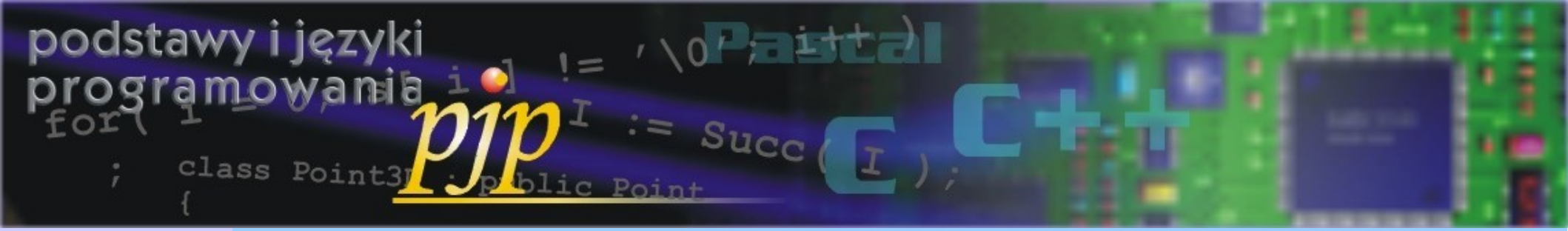


Podstawy programowania w języku C dla środowiska Windows

Część druga

Windows API — pierwsze kroki

Autor
Roman Simiński

Kontakt
roman.siminski@us.edu.pl
www.us.edu.pl/~siminski

Obsługa wybranych komunikatów — WM_CREATE

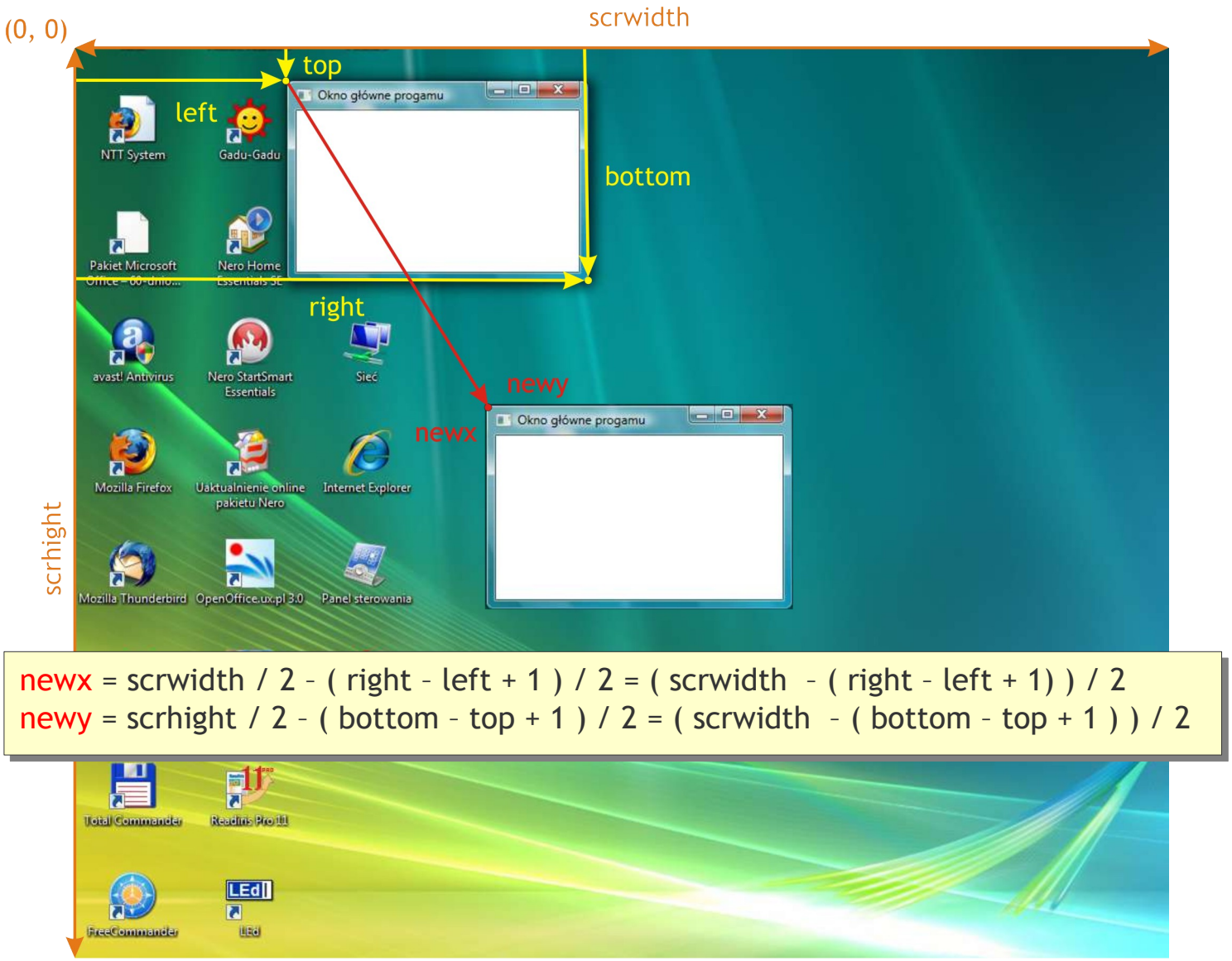
```
LRESULT CALLBACK WndProc01( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
  switch( uMsg)
  {
    case WM_CREATE:
      MessageBox( NULL, "Uwaga - rodzi się okno", "Info", MB_OK );
      break;

    case WM_CLOSE:
      DestroyWindow( hWnd );
      break;

    case WM_DESTROY:
      PostQuitMessage( 0 );
      break;

    default:
      return DefWindowProc( hWnd, uMsg, wParam, lParam);
  }
  return 0;
}
```

Obsługa komunikatu generowanego w trakcie tworzenia okna

- Komunikat WM_CREATE jest wysyłany podczas wykonania *CreateWindowEx* lub *CreateWindow*. Procedura obsługi komunikatów *nowego* okna otrzymuje ten komunikat *po utworzeniu* okna ale *przed* jego *wyświetleniem*.

Obsługa wybranych komunikatów — WM_CREATE, przykład

Centrowanie okna względem ekranu:



newx = scrwidth / 2 - (right - left + 1) / 2 = (scrwidth - (right - left + 1)) / 2
newy = scrhight / 2 - (bottom - top + 1) / 2 = (scrwidth - (bottom - top + 1)) / 2

Obsługa wybranych komunikatów — WM_CREATE, przykład

Potrzebujemy:

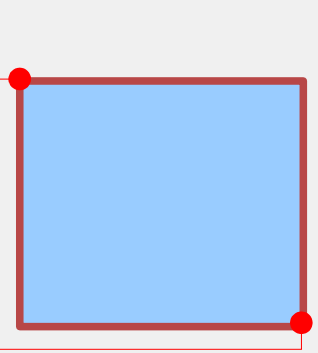
- *wysokości* i *szerokości* ekranu,
- *wysokości* i *szerokości* okna, które można wyznaczyć na podstawie *współrzędnych* przeciwległych narożników *prostokąta* opisującego okno,
- funkcji pozwalającej na ustawienie okna na zadaj pozycji ekranowej,
- wywołanie tej funkcji umieścimy w obsłudze komunikatu WM_CREATE.

Obsługa wybranych komunikatów — WM_CREATE, przykład

```
RECT rc;
. . .

case WM_CREATE:

        . . .



       break;
```

Rekord opisu prostokątnego obszaru ekranowego

```
typedef struct _RECT
{
  LONG left;
  LONG top;
  LONG right;
  LONG bottom;
} RECT, *PRECT;
```



- RECT — określa współrzędne *lewego górnego* i prawego dolnego rogu prostokąta.
- Uwaga — zgodnie z konwencją prostokąt opisuje zewnętrzny obrys obszaru, tzn. piksele opisujące narożniki nie wchodzą w skład rozważanego obszaru prostokątnego.

Obsługa wybranych komunikatów — WM_CREATE, przykład

```
RECT rc;
. . .

case WM_CREATE:
      GetWindowRect( hWnd, &rc );
       . . .


      break;
```

Pobranie informacji o położeniu okna hWnd do rekordu rc

- `BOOL GetWindowRect( HWND hWnd,  LPRECT lpRect )` — pobiera rozmiary prostokąta w który jest wpisane okno identyfikowane pierwszym parametrem i zapisuje te informacje do rekordu opisu prostokąta, wskazywanego prze drugi parametr.
- Współrzędne są liczone w współrzędnych ekranowych odniesionych do lewego górnego narożnika ekranu.left corner of the screen.

Obsługa wybranych komunikatów — WM_CREATE, przykład

```
RECT rc ;
. . .

case WM_CREATE:

      GetWindowRect( hWnd, &rc );

      SetWindowPos( hWnd, 0,
        ( GetSystemMetrics( SM_CXSCREEN ) - ( rc.right - rc.left + 1 ) ) / 2,
        ( GetSystemMetrics( SM_CYSCREEN ) - ( rc.bottom - rc.top + 1 ) ) / 2,
         0, 0, SWP_NOZORDER|SWP_NOSIZE );

      break;
```

Ustalenie położenia okna centralnie względem ekranu

- `BOOL SetWindowPos( HWND hWnd, HWND hWndInsertAfter, int X, int Y, int cx, int cy, UINT uFlags);`
- Zmienia pozycję, rozmiar oraz kolejność okna na osi Z.
- `GetSystemMetrics` — pobiera określone informacje o rozmiarach i ustawieniach systemu liczone w pikselach. Parametry `SM_CXSCREEN` i `SM_CYSCREEN` pozwalają na pobranie szerokości i wysokości ekranu podstawowego wyświetlacza.

Obsługa wybranych komunikatów — WM_CREATE, przykład

- `BOOL SetWindowPos( HWND hWnd, HWND hWndInsertAfter, int X, int Y, int cx, int cy, UINT uFlags );`
- `hWnd` — identyfikator pozycjonowanego okna.
- `hWndInsertAfter` — identyfikator okna poprzedzającego na osi Z, lub HWND_BOTTOM, HWND_NOTOPMOST, HWND_TOP, HWND_TOPMOST,
- `X, Y` — nowa pozycja x i y lewego górnego rogu,
- `cx, cy` — nowa szerokość i wysokość okna liczona w pikselach,
- `uFlags` — Określa sposób zmiany rozmiaru i pozycjonowania okna, wiele możliwości np. kombinacja: `SWP_NOSIZE | SWP_NOZORDER` zachowuje aktualną kolejność na osi Z i bieżacy rozmiar (parametry cx i cy są ignorowane).

Obsługa wybranych komunikatów – WM_CREATE, ładniejszy przykład

```
void CenterWindow( HWND hWnd )
{
    RECT rc ;

    GetWindowRect ( hWnd, &rc ) ;

    SetWindowPos( hWnd, 0,
        ( GetSystemMetrics( SM_CXSCREEN ) – ( rc.right - rc.left + 1 ) ) / 2,
        ( GetSystemMetrics( SM_CYSCREEN ) – ( rc.bottom - rc.top + 1 ) ) / 2,
        0, 0, SWP_NOZORDER|SWP_NOSIZE );
}


LRESULT CALLBACK WndProc01( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
  switch( uMsg)
  {
    case WM_CREATE:
      CenterWindow( hWnd );
      break;
  . . .
  }
}
```

Obsługa wybranych komunikatów — WM_KEYDOWN

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:

    . . .
    break;
 . . .
}
. . .
```

Pobranie informacji o położeniu okna hWnd do rekordu rc

- `WM_KEYDOWN` — ten komunikat jest wstawiany do kolejki gdy naciśnięto niesystemowy klawisza wirtualnego (bez Alt). Ten komunikat ma dwa parametry, `wParam` określa tzw. kod naciśniętego klawisza, `lParam` określa dodatkowe informacje związane z naciśniętym klawiszem.
- Klawisz wirtualny (ang. *virtual-key*) ma swój niezależny od platformy identyfikator (np. VK_ESCAPE, VK_SPACE, VK_PRIOR (PageUp), VK_NEXT (PageDown), VK_END, VK_HOME, VK_LEFT, VK_UP, VK_RIGHT, VK_DOWN, VK_RETURN (Enter)).

Jak zamknąć okno naciskając klawisz Escape?

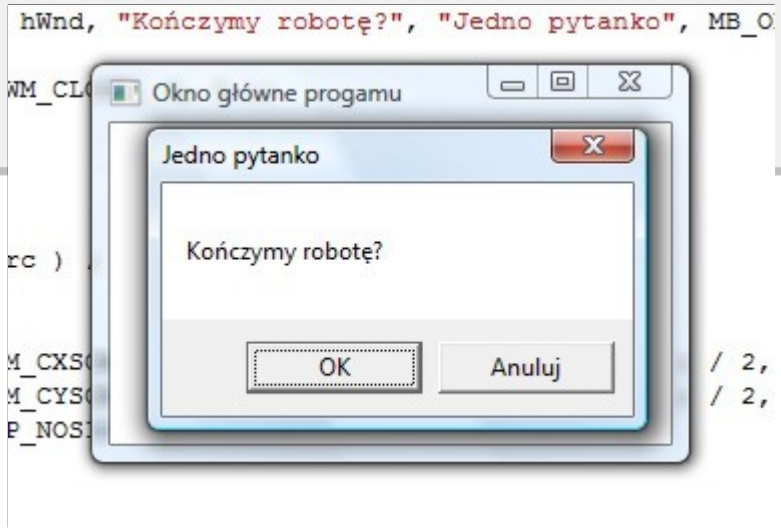
```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
    if( wParam == VK_ESCAPE )
    {
      . . .
    }
    break;
 . . .
}
. . .
```

Identyfikacja klawisza

Jak zamknąć okno naciskając klawisz Escape?

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
    if( wParam == VK_ESCAPE )
    {
      int ret = MessageBox( NULL, "Kończymy robotę?", "Jedno pytanko", MB_OKCANCEL );
      if ( ret == IDOK )
         . . .;
    }
    break;
 . . .
}
. . .
```

Spytaj czy kończymy



Jak zamknąć okno naciskając klawisz Escape?

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
   if( wParam == VK_ESCAPE )
   {
     int ret = MessageBox( NULL, "Kończymy robotę?", "Jedno pytanko", MB_OKCANCEL );
     if ( ret == IDOK )
       SendMessage( hWnd, WM_CLOSE, 0, 0 );
   }
   break;
 . . .
}
. . .
```

Wyślij komunikat WM_CLOSE

- `LRESULT SendMessage( HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam );`
- `SendMessage` — funkcja *wysyła* komunkat określony paramtrami `Msg` i `wParam`, `wParam` do okna określonego `hWnd`. Funkcja wywołuje procedurę obsługi komunikatów okna i czeka na zakończenie jej wywołania (czyli aż obsłużony zostanie komunikat).

Jak przechwycić klawisz Enter?

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
   if( wParam == VK_RETURN )
      MessageBox( NULL, "Poco wciskasz ten Enter!", "Ostrzeżenie", MB_OK );
   break;
 . . .
}
. . .
```

Identyfikacja klawisz Enter

Rozpoznawania większej liczby klawiszy

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
    switch( wParam )
    {
      case VK_RETURN :
        MessageBox( NULL, "Poco wciskasz ten Enter!", "Ostrzeżenie", MB_OK );
        break;

      case VK_ESCAPE :
        MessageBox( NULL, "Okno zamyka Alt-F4", "Ostrzeżenie", MB_OK );
        break;
    }
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

Taki sobie dziwny przykład

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
    switch( wParam )
      if( wParam == VK_SCROLL )
      {
        char szFileName[ MAX_PATH ];
        HINSTANCE hInstance = GetModuleHandle( NULL );

        GetModuleFileName( hInstance, szFileName, MAX_PATH );
        MessageBox( hWnd, szFileName, "To program z pliku", MB_OK );
      }
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

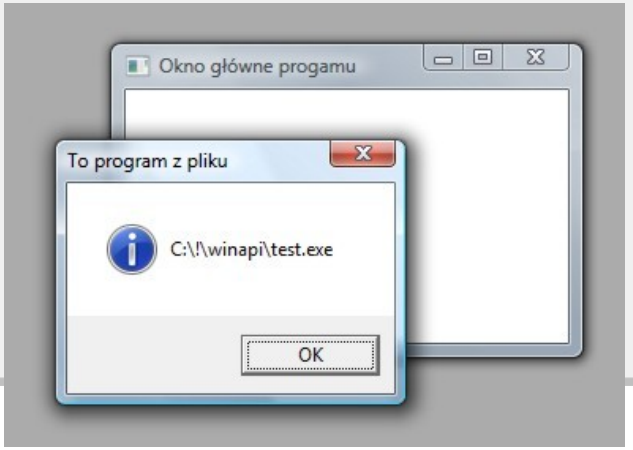
Pobranie identyfikatora aktualnie działającego modułu

- `GetModuleHandle(NULL)` — te szczególny przypadek wywołania funkcji `GetModuleHandle` pozwala na pobranie identyfikatora własnego procesu.

Taki sobie dziwny przykład

```
. . .
switch( uMsg )
{
  case WM_KEYDOWN:
    switch( wParam )
      if( wParam == VK_SCROLL )
      {
        char szFileName[ MAX_PATH ];
        HINSTANCE hInstance = GetModuleHandle( NULL );

        GetModuleFileName( hInstance, szFileName, MAX_PATH );
        MessageBox( hWnd, szFileName, "To program z pliku", MB_OK );
      }
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

Pobierz i wyświetl nazwę pliku, z którego został uruchomiony aktualnie działający kod



- `GetModuleHandle(NULL)` — pobranie pełnej nazwy pliku zawierającego moduł określony pierwszym parametrem wywołania do tablicy identyfikowanej drugim parametrem (nie więcej znaków niż wskazuje ostatni parametr).

Identyfikowanie komunikatów pochodzących od myszy

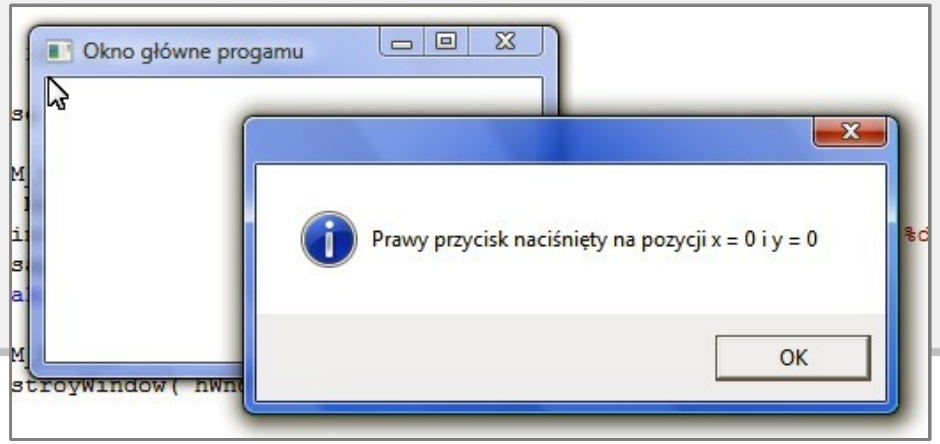
```
. . .
switch( uMsg )
{
  case WM_LBUTTONDOWN:
    MessageBox( hWnd, "Lewy przycisk kliknięty", "", MB_OK | MB_ICONINFORMATION);
    break;

  case WM_RBUTTONDOWN:
    MessageBox( hWnd, "Prawy przycisk kliknięty", "", MB_OK | MB_ICONINFORMATION);
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

Identyfikowanie naciśniętego klawisza myszy

Identyfikacja pozycji wskaźnika myszy względem obszaru klienta

```
. . .
POINTS p;
char mess[ 128 ];

switch( uMsg )
{
  case WM_LBUTTONDOWN:

    p = MAKEPOINTS( lParam );

    sprintf( mess, "Prawy przycisk naciśnięty na pozycji x = %d i y = %d", p.x, p.y );
    MessageBox( hWnd, mess, "", MB_OK | MB_ICONINFORMATION);
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

Parametr lParam zawiera pozycje punktu centralnego wskaźnika myszy w trakcie zajścia zdarzenia. Młodszy bajt zawiera współrzędną x, starszy zawiera współrzędną y.



- Współrzędne zapisane w parametrze *lParam* wyrażone są w układzie współrzędnych obszaru klienta — (0, 0) znajduje się w lewym górnym rogu obszaru klienckiego okna.

Identyfikacja pozycji wskaźnika myszy względem obszaru klienta

```
. . .
POINTS p;
char mess[ 128 ];

switch( uMsg )
{
  case WM_LBUTTONDOWN:

    p = MAKEPOINTS( lParam );

     sprintf( mess, "Prawy przycisk naciśnięty na pozycji x = %d i y = %d", p.x, p.y );
    MessageBox( hWnd, mess, "", MB_OK | MB_ICONINFORMATION);
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
    break;
  . . .
}
. . .
```

Makro MAKEPOINTS wydobywa ze parametru lParam współrzędne centralnego punktu wskaźnika myszy i tworzy rekord opisu punkty typu POINTS.

- Parametr *wParam* opisuje stan klawiszy *Ctrl* i *Shift* oraz pozostałych klawiszy myszy w trakcie zaistnienia zdarzenia — to czasami jest istotne.

Identyfikacja podwójnego kliknięcia

```
. . .
POINTS p;
char mess[ 128 ];

switch( uMsg )
{
  case WM_LBUTTONDBLCLK:

    p = MAKEPOINTS( lParam );

    sprintf( mess, "Prawy przycisk naciśnięty na pozycji x = %d i y = %d", p.x, p.y );
    MessageBox( hWnd, mess, "", MB_OK | MB_ICONINFORMATION);
    break;

  case WM_CLOSE:
    DestroyWindow( hWnd );
```

Aby ten komunikat dotarł do procedury obsługi komunikatów, okno musi zostać zdefiniowane w wykorzystaniem stylu CS_DBLCLKS

```
int WINAPI WinMain( HINSTANCE hInst, HINSTANCE hPrev, LPSTR lpszCmdLine, int nCmdShow )
{
  WNDCLASSEX winClass;
  HWND hWndMain;

  winClass.cbSize        = sizeof( WNDCLASSEX );
  winClass.cbClsExtra    = 0;
  winClass.style         = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS ;
  . . .
}
```

Komunikat WM_PAINT — informacje wprowadzające

- Operacje graficzne realizowane w ramach okna odbywają się z wykorzystaniem tzw. *kontekstu urządzania graficznego* (ang. *device context*), zwanego dalej DC.
- Intuicyjnie DC jest to narzędzie pozwalające na realizację operacji graficznych w ten sam sposób, niezależnie od „płótna” na jakim się to odbywa i niezależnie od specyficznych jego właściwości.
- Formalnie DC to struktura definiująca zbiór graficznych obiektów oraz związanych z nimi atrybutów — obejmują one *pióra* do kreślenia linii, *pędzle* do malowania i wypełniania obszarów, mapy bitowe, palety kolorów, oraz inne elementy istotne elementy wykonywania operacji graficznych.

Komunikat WM_PAINT — informacje wprowadzające

Wyróżnia się cztery typy DC:

- *Display* — przeznaczony do realizacji operacji na ekranie.
- *Printer* — przeznaczony do realizacji operacji na drukarce lub ploterze.
- *Memory* — przeznaczony do realizacji operacji na mapach bitowych.
- *Information* — przeznaczony do pobierania informacji o danych DC.

Przy okazji — dwa istotne pojęcia:

- *obszar obcinania* (ang. *clipping region*) — wszystkie operacje wykonywane przez aktualnie stosowany obiekt graficzny realizowane są tylko w ustalonym obszarze obcinania.
- *obszar aktualizacji* (ang. *update region*) — obszar określający tę część okna, która wymaga aktualizacji. Jeżeli system wykryje konieczność aktualizacji przynajmniej części okna, ustali odpowiedni obszar aktualizacji w wyśle komunikat WM_PAINT.

Komunikat WM_PAINT — informacje wprowadzające

Kontekst wyświetlacza — DDC (ang. *display device contexts*)

- Program otrzymuje kontekst DDC poprzez wywołanie funkcji *BeginPaint*, *GetDC* lub *GetDCEx*.
- Zwykle wykorzystuje się otrzymany DDC do wykonywania operacji graficznych w obszarze klienckim okna.
- Po zakończeniu operacji graficznych należy wywołać funkcje *EndPaint* or *ReleaseDC*.

Komunikat WM_PAINT — pobieranie kontekstu wyświetlacza

- Funkcja *BeginPaint* jest dedykowana dla realizacji operacji graficznych podczas obsługi komunikatu WM_PAINT.
- Funkcja *BeginPaint* ustala *obszar obcinania* na taki jak *obszar aktualizacji*.
- Każda operacja graficzna wykonana po *BeginPaint* będzie od razu widoczna na ekranie.
- Funkcja *BeginPaint* wysyła komunikat *WM_ERASEBKGND*, który spowoduje wyczyszczenie obszaru klienckiego przy użyciu pędzla określonego w zarejestrowanej klasie okna.
- Funkcja *BeginPaint* wypełnia danymi rekord *PAINTSTRUCT*, jej rezulataem jest identyfikator *DDC* lub *NULL* gdy nie jest on dostępny.
- Po zakończeniu operacji graficznych należy koniecznie wywołać funkcje *EndPaint*.

Komunikat WM_PAINT – wywołanie BeginPaint

```
. . .
PAINTSTRUCT ps;
HDC hdc;

switch( uMsg )
{
  case WM_PAINT:

       hdc = BeginPaint( hWnd, &ps );

       TextOut( hdc, 0, 0, "Dzień dobry", 11 );

       EndPaint( hWnd, &ps );

       break;
 . . .
```

Definicja rekordu PAINTSTRUCT – zawiera on informacje wykorzystywane do rysowania w obszarze klienckim okna aplikacji.
HDC – identyfikator DDC

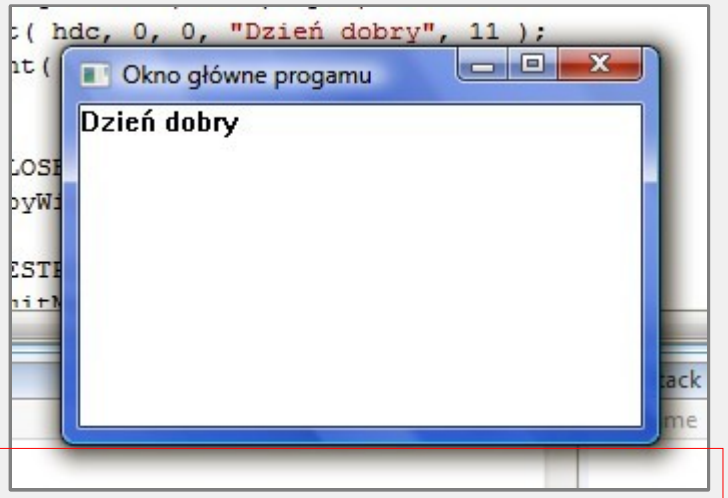
Komunikat WM_PAINT — wywołanie BeginPaint

```
. . .
PAINTSTRUCT ps;
HDC hdc;

switch( uMsg )
{
  case WM_PAINT:

       hdc = BeginPaint( hWnd, &ps );

       TextOut( hdc, 0, 0, "Dzień dobry", 11 );

       EndPaint( hWnd, &ps );

       break;
 . . .
```

Pobranie kontekstu urządzenia — zapamiętanie w *hdc* — i wstawienie informacji do rekordu *ps*

Komunikat WM_PAINT — wywołanie TextOut

```
. . .
PAINTSTRUCT ps;
HDC hdc;

switch( uMsg )
{
   case WM_PAINT:

        hdc = BeginPaint( hWnd, &ps );

        TextOut( hdc, 0, 0, "Dzień dobry", 11 );

        EndPaint( hWnd, &ps );

        break;
 . . .
```



Wykonanie przykładowej operacji graficznej — narysowanie na *hdc*, na pozycji x = 0 oraz y = 0, tekstu ”Dzień dobry”, liczącego 11 znaków.
OutText wykorzystuje font, kolor pierwszego planu i tła ustawiony dla wykorzystywanego kontekstu identyfikowanego przez parametr *hdc*.

Komunikat WM_PAINT – wywołanie EndPaint

```
. . .
PAINTSTRUCT ps;
HDC hdc;

switch( uMsg )
{
  case WM_PAINT:

       hdc = BeginPaint( hWnd, &ps );

       TextOut( hdc, 0, 0, "Dzień dobry", 11 );

       EndPaint( hWnd, &ps );

       break;
 . . .
```

Funkcja *EndPaint* sygnalizuje, że operacja rysowania w oknie identyfikowanym przez hWnd została zakończona.