Graphics Device Interface (GDI – interfejs urządzenia graficznego) jest podsystemem Windows odpowiedzialnym za wyświetlanie grafiki (włącznie z napisami) na ekranie i wydrukach.

Kolory w Windows są pamiętane w 32 bitowych liczbach całkowitych bez znaku (unsigned long) Opisujący je typ danych ma nazwę COLORREF. Wartość tego typu zwraca makro RGB, które pobiera trzy argumenty reprezentujące barwę czerwoną, zieloną i niebieską:
```
COLORREF RGB(BYTE byRed, BYTE byGreen, BYTE byBlue);
```
Makrodefinicje: `GetRValue, GetGValue, GetBValue` pozwalają obliczyć z wartości koloru RGB wartość odpowiednich barw podstawowych (0-255).

Przy omawianiu komunikatów poznaliśmy dwa podstawowe sposoby pobrania uchwytu kontekstu urządzenia.
Gdy program otrzymuje HDC, Windows tworzy kontekst urządzenia przyjmując wartości domyślne dla każdego jego atrybutu. W programie każdą z tych wartości można odczytać lub zmienić.

Atrybut DC	***Wartość domyślna***	***Funkcja zmieniająca wartość***	***Funkcja zwracająca wartość***
Tryb odwzorowania *(mapping mode)*	MM_TEXT	*SetMapMode*	*GetMapMode*
Punkt początkowy okna (*window origin*)	(0, 0)	*SetWindowOrgEx* *OffsetWindowOrgEx*	*GetWindowOrgEx*
Punkt początkowy widoku (*viewport origin*)	(0, 0)	*SetViewportOrgEx* *OffsetViewportOrgEx*	*GetViewportOrgEx*
Rozciągłość okna *(window extents)*	(1, 1)	*SetWindowExtEx* *SetMapMode* *ScaleWindowExtEx*	*GetWindowExtEx*
Rozciągłość widoku *(viewport extents)*	(1, 1)	*SetViewportExtEx* *SetMapMode* *ScaleViewportExtEx*	*GetViewportExtEx*
Pióro (*pen*)	BLACK_PEN	*SelectObject*	*SelectObject*
Pędzel *(brush)*	WHITE_BRUSH	*SelectObject*	*SelectObject*
Czcionka *(font)*	SYSTEM_FONT	*SelectObject*	*SelectObject*
Mapa bitowa *(bitmap)*	Brak	*SelectObject*	*SelectObject*
Bieżąca pozycja pióra	(0, 0)	*MoveToEx* *LineTo* *PolylineTo* *PolyBezierTo*	*GetCurrentPositionEx*
Tryb malowania tła (background mode)	OPAQUE	*SetBkMode*	*GetBkMode*
Kolor tła *(background color)*	Biały	*SetBkColor*	*GetBkColor*
Kolor napisów *(text color)*	Czarny	*SetTextColor*	*GetTextColor*
Tryb kreślenia *(drawing mode)*	R2_COPYPEN	*SetROP2*	*GetROP2*
Tryb rozciągania mapy bitowej *(stretching mode)*	BLACKONWHITE	*SetStretchBltMode*	*GetStretchBltMode*
Tryb wypełniania wielokątów *(polygon fill mode)*	ALTERNATE	*SetPolyFillMode*	*GetPolyFillMode*
Odstęp między znakami	0	*SetTextCharacterExtra*	*GetTextCharacterExtra*
Punkt początkowy pędzla *(Brush Origin)*	(0, 0)	*SetBrushOrgEx*	*GetBrushOrgEx*
Region obcinania *(clipping region)*	Brak	*SelectObject* *SelectClipRgn* *IntersectClipRgn* *OffsetClipRgn* *ExcludeClipRect* *SelectClipPath*	*GetClipBox*

W momencie gdy zwalniamy DC funkcjami EndPaint, ReleaseDC wszelkie zmiany atrybutów DC są tracone. Możemy je zapamiętać określając w stylach klasy:
CS_OWNDC –każde okno danej klasy będzie miało swój własny DC. Kontekst ten będzie istnieć dopóki nie usuniemy okna, a jego atrybuty pozostaną stałe, dopóki się ich jawnie nie zmieni.
lub:
CS_CLASSDC – dana klasa ma własny kontekst urządzenia dzielony między wszystkie okna tej klasy.

Czasami pojawia się konieczność zmiany atrybutów na krótka chwile i ponowne przywrócenie poprzednich atrybutów. Dokonamy tego stosując funkcje:

```
  int iSaveID = SaveDC(hdc);
  // zmiana atrybutów i działania na zmienionym DC
  RestoreDC(hdc, -1);
```

```
BOOL RestoreDC(HDC hdc, int nSavedDC)
```

nSavedDC – wskazuje na stan DC do odtworzenia. Jeżeli >0 wskazuje na wartość zwróconą przez SaveDC, jeżeli jest to wartość ujemna wskazuje który zapisany DC ma odtworzyć np. –1 spowoduje odtworzenie ostatnio zapisanego DC.
W trakcie rysowania przyda się funkcja którą poznaliśmy wcześniej (ćw.2) zwracająca 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 odczytująca parametry systemu:

```
int GetSystemMetrics(int nIndex);
```

nIndex - indeks parametru, którego wartość chcemy otrzymać, np. SM_CXSCREEN zwraca szerokość ekranu
funkcja odczytująca lub ustawiająca zadany parametr

```
BOOL SystemParametersInfo(UINT uiAction, UINT uiParam,
                          PVOID pvParam,UINT fWinIni);
```

`uiAction` - jaki parametr należy odczytać lub ustawić
`uiParam, pvParam` - parametry zależne od podejmowanej akcji
`fWinIni` - czy aktualizować ustawienia (rozesłanie komunikatu: WM_SETTINGCHANGE)
np. `SystemParametersInfo(SPI_GETWORKAREA,0,&rect,0)` pobiera rozmiar obszaru roboczego (bez paska zadań)

Tryb odwzorowania.

Większość współrzędnych i wymiarów używanych we wszystkich funkcjach GDI jest wyrażone w jednostkach logicznych. Windows zmienia jednostki logiczne na jednostki urządzenia, czyli na piksele. Sposób zmiany współrzędnych zależy m.in. od trybu odwzorowania.
Windows wyróżnia osiem trybów odwzorowywania:

		Kierunek wzrostu wartości	
Tryb odwzorowania	*Jednostki logiczne*	*na osi x*	*na osi y*
`MM_TEXT` (domyślny)	piksel	w prawo	w dół
`MM_LOMETRIC`	`0,1 mm`	w prawo	do góry
`MM_HIMETRIC`	`0,01 mm`	w prawo	do góry
`MM_LOENGLISH`	`0,01 cala`	w prawo	do góry
`MM_HIENGLISH`	`0,001 cala`	w prawo	do góry
`MM_TWIPS`	`1/1440 cala`	w prawo	do góry
`MM_ISOTROPIC`	dowolne `(x=y)`	do wyboru	do wyboru - można zmienić rozciągłość okna/widoku
`MM_ANISOTROPIC`	dowolne (x!=y)	do wyboru	do wyboru - można zmienić rozciągłość okna/widoku

`int` **`SetMapMode`**`(HDC hdc, int iMapMode)` – pozwala na zmianę trybu odwzorowania..
`int` **`GetMapMode`**`(HDC hdc)` – zwraca tryb odwzorowania.
Tryb odwzorowania określa odwzorowanie „okna” (współrzędne logiczne) na „widok” (współrzędne urządzenia) za pomocą:

```
xViewport =(xWindow – xWindOrg) * (xViewExt / xWinExt) +
xViewOrg
yViewport =(yWindow – yWindOrg) * (yViewExt / yWinExt) +
yViewOrg
```

xWindow, yWindow – przekształcany punkt we współrzędnych logicznych
xViewport, yViewport – wynik przekształcenia we współrzędnych urządzenia.
xWindOrg, yWindOrg, xViewOrg, yViewOrg – początek okna i widoku
xViewExt, yViewExt, xWinExt, yWindExt – rozciągłość widoku i okna. Każda rozciągłość z osobna nic nie znaczy, ale stosunek rozciągłości widoku do rozciągłości okna określa czynnik skalujący niezbędny do przeliczania.
`BOOL` **`DPtoLP`**`(HDC hdc, LPPOINT lpPoints, int iNumber)` – funkcja przekształca iNumber punktów urządzenia zawartych w tablic lpPoints na punkty logiczne.
`BOOL` **`LPtoDP`**`(HDC hdc, LPPOINT lpPoints, int iNumber)` – funkcja przekształca iNumber punktów logicznych zawartych w tablic lpPoints na punkty urządzenia.

Jeżeli chcesz wyświetlić obraz w calach lub milimetrach bez zmiany trybu można dzięki funkcji *GetDeviceCaps* uzyskać potrzebne informacje i samemu odpowiednio wyskalować rysunek.
`int` **`GetDeviceCaps`**`(HDC hdc, int nIndex)` – zwraca informacje na temat DC.
nIndex – indeks danej o którą pytamy
HORZSIZE – szerokość w mm
VERTSIZE – wysokość w mm
HORZRES – szerokość w pikselach
VERTRES – wysokość w liniach rastra
BITSPIXEL – liczba bitów na piksel (głębia koloru)
PLANES – liczba warstw koloru
NUMBRUSHES – liczba pędzli urządzenia
NUMPENS – liczba piór urządzenia
NUMMARKERS – liczba znaczników urządzenia
NUMFONTS – liczba czcionek urządzenia
NUMCOLORS – liczba kolorów urządzenia
PDEVICESIZE – rozmiar struktury urządzenia
ASPECTX – względna szerokość piksela
ASPECTY – względna wysokość piksela
ASPECTXY – względna przekątna piksela
LOGPIXELSX – punkty na cal w poziomie
LOGPIXELSY – punkty na cal w pionie
SIZEPALETTE – liczba pozycji palety
NUMRESERVED – zarezerwowane pozycje palety
COLORRES – faktyczna rozdzielczość koloru
`BOOL` **`ClientToScreen`**`(HWND hwnd, LPPOINT lpPoint)` – przekształca współrzędne wskazane przez lpPoint obszaru roboczego na współrzędne ekranu.
`BOOL` **`ScreenToClient`**`(HWND hwnd, LPPOINT lpPoint)` – przekształca współrzędne ekranu na współrzędne obszaru roboczego.

Pióro.

Do rysowania w kontekście urządzenia używamy „pióra”. Pióro określa kolor linii, jej grubość oraz styl (ciągła, kropkowana, przerywana).
W systemie mamy trzy predefiniowane pióra:
BLACK_PEN – domyślny, czarne ciągłe linie grubości jednego piksela.
WHITE_PEN
NULL_PEN – pióro jest przeźroczyste.
Oby otrzymać w programie predefiniowane pióro wykorzystujemy funkcje *GetStockObject*.

```
HPEN hPen = GetStockObject(WHITE_PEN);
```

Aby dane pióro było bieżące w kontekście urządzenia korzystamy z funkcji *SelectObject*.

```
HPEN hPenOld = SelectObject(hdc, hPen);
```

`HGDIOBJ` **`SelectObject`**`(HDC hdc, HGDIOBJ hGDIobj)` – funkcja ustawia obiekt GDI w określonym kontekście urządzenia i zwraca uchwyt obiektu, który został zastąpiony. Obiektem może być bitmapa, czcionka, pióro, pędzel.

`HPEN` **`CreatePen`**`(int PenStyle, int iWidth, COLORREF rgbColor)` - tworzy „pióro logiczne” które może następnie być wybrane w DC przy pomocy *SelectObject.*
PenStyle – określa styl linii:
PS_SOLID _____
PS_DASH __ __ __ __ __
PS_DOT .....
PS_DASHDOT _._._._._._
PS_DASHDOTDOT _.._.._.._.._
PS_NULL
PS_INSIDEFRAME _____ - odpowiada PS_SOLID. W przypadku prostokąta jego wymiary są zmniejszane tak aby ramka mieściła się wewnątrz podanych wymiarów.
iWidth – dla PS_SOLID, PS_NULL i PS_INSIDEFRAME grubość linii. 0 oznacza jeden piksel.

`HPEN` **`CreatePenIndirect`**`( CONST LOGPEN * logpen)` - tworzy „pióro logiczne” które może następnie być wybrane w DC przy pomocy *SelectObject,* przy pomocy struktury **LOGPEN**.

```
typedef struct tagLOGPEN {
UINT PenStyle;
POINT iWidth;            // system ignoruje iWidth.y i bierze pod uwagę jedynie iWidth.x
COLORREF rgbColor;
} LOGPEN, *PLOGPEN;
```

`HPEN` **`ExtCreatePen(`**`DWORD dwPenStyle, DWORD dwWidth, CONST LOGBRUSH * lplb, DWORD dwStyleCount, CONST DWORD *lpStyle)` – tworzy rozszerzone pióro. Część jego możliwości nie jest wykorzystywana przy kreśleniu zwykłych linii. Więcej jego właściwości wykorzystywane jest przy kreśleniu ścieżki za pomocą *StrokePath*.
dwPenStyle – określa styl linii. Można połączyć style używane w *CreatePen* z stylami:
PS_GEOMETRIC – dwWidth określa grubość linii w jednostkach logicznych
PS_COSMETIC – dwWidth musi się równać 1.
PS_ENDCAP_ROUND – zaokrąglone końce linii (domyślne w *CreatePen*)

PS_ENDCAP_SQUARE – kończy linię kwadratem wydłużając linię o pół grubości linii na każdym jej końcu.
PS_ENDCAP_FLAT – kończy linię kwadratem
PS_JOIN_ROUND – połączenia między liniami są zaokrąglane (domyślne w *CreatePen*)
PS_JOIN_BEVEL – ścina końce połączenia
PS_JOIN_MITER – tworzy połączenie spiczaste
lplp – wskaźnik na strukturę opisującą pędzel. Jeżeli styl jest PS_COSMETIC pole struktury lbStyle = BS_SOLID, a lbColor określa kolor. Jeżeli PS_GEOMETRIC – dowolne.
Można pobrać informacje o piórze:

```
GetObject(hPen, sizeof(LOGPEN), (LPVOID) &logpen);
```

`int` **`GetObject`**`(HGDIOBJ hGDIobj, int cbBuffor, LPVOID lpvObj)` – funkcja zwraca informacje na temat obiektu GDI wypełniając bufor przechowujący informacje na temat obiektu.
cbBuffor – określa wielkość bufora.

Wszystkie utworzone przez użytkownika obiekty GDI muszą być usunięte.

```
BOOL DeleteObject(HGDIOBJ hGDIobj);
```

Nie wolno usuwać obiektów GDI w chwili gdy są wybrane w nie zwolnionym DC, oraz obiektów predefiniowanych.
Przerwy w liniach, tło tekstu i pędzla jest wypełniane kolorem tła DC jeżeli tryb tła jest ustawiony na OPAQUE. Jeżeli tryb tła jest TRANSPARENT wówczas Windows pozostawia tło bez wypełnienia.
`int` **`SetBkMode`**`(HDC hdc, int iBkMode)` – ustawia tryb tła (OPAQUE, TRANSPARENT)
`int` **`GetBkMode`**`(HDC hdc)` – zwraca tryb tła (OPAQUE, TRANSPARENT)
`int` **`SetBkColor`**`(HDC hdc, COLORREF rgbColor)` – ustawia kolor tła
`COLORREF` **`GetBkColor`**`(HDC hdc)` – zwraca kolor tła.

Tryb rysowania określa wynik połączenia pikseli pióra i podłoża – kolor linii jest wynikiem działań logicznych na kolorze pióra i podłoża.

R2_BLACK	– `0`	Piksel jest zawsze czarny
R2_NOTMERGEPEN	– `~(P|T)`	Piksel jest inwersją koloru uzyskanego trybem R2_MERGEPEN
R2_MASKNOTPEN	– `~P &T`	
R2_NOTCOPYPEN	– `~P`	Piksel jest inwersją koloru pióra
R2_MASKPENNOT	– `P& ~T`	
R2_NOT	– `~T`	Piksel jest inwersją koloru tła
R2_XORPEN	– `P^T`	
R2_NOTMASKPEN	– `~(P&T)`	
R2_MASKPEN	– `P&T`	
R2_NOTXORPEN	– `~(P^T)`	
R2_NOP	– `T`	Piksel pozostaje niezmieniony
R2_MERGENOTPEN	– `~P|T`	
R2_COPYPEN	– `P`	Domyślny. Piksel jest koloru pióra
R2_MERGEPENNOT	– `P| ~T`	
R2_MERGEPEN	– `P|T`	Piksel jest kombinacją koloru tła i pióra
R2_WHITE	– `1`	Piksel jest zawsze biały

`int` **`SetROP2`**`(HDC hdc, int iDrawMode)` – ustawia tryb rysowania.
`int` **`GetROP2`**`(HDC hdc)` – zwraca tryb rysowania.

Pędzel.

Do wypełniania obszarów używamy „pędzla”. Pędzel jest mapą bitową o wymiarach 8 pikseli na 8 pikseli.
W systemie mamy predefiniowane pędzle:
WHITE_BRUSH – domyślny, wypełnia obszar na biało.
LTGRAY_BRUSH
GRAY_BRUSH
DKGRAY_BRUSH
BLACK_BRUSH
NULL_BRUSH = HOLLOW_BRUSH
Można wybrać predefiniowany pędzel tak jak wybierało się predefiniowane pióro:

```
HBRUSH hBrush = GetStockObject(GRAY_BRUSH);
HBRUSH hBrushOld = SelectObject(hdc, hBrush);
```

`HBRUSH` **`CreateSolidBrush`**`(COLORREF rgbColor)` – tworzy pędzel logiczny o określonym kolorze.
`HBRUSH` **`CreateHatchBrush`**`(int iHatchStyle, COLORREF rgbColor)` – tworzy pędzel ze wzorem o określonym kolorze.
iHatchStyle – określa rodzaj wzoru:
HS_HORIZONIAL 
HS_VERICAL 
HS_FDIAGONAL 
HS_BDIAGONAL 
HS_CROSS 
HS_DIAGCROSS 
Obszar między liniami jest wypełniany w zależności od koloru i trybu tła (jak pióro z przerywanymi liniami).
`HBRUSH` **`CreatePatternBrush`**`(HBITMAP hBitmap)` – tworzy pędzel w oparciu o mapę bitową.
`HBRUSH` **`CreateBrushIndirect`**`( CONST LOGBRUSH * logbrush)` – tworzy pędzel w oparciu o strukturę **LOGBRUSH**. Struktura składa się z trzech pól. Wartość pola lbStyle określa w jaki sposób Windows interpretuje pozostałe dwa pola

LbSytle (UINT)	*lbColor (COLORREF)*	*lbHatch (LONG)*
BS_SOLID	kolor pędzla	pomijane
BS_HOLLOW	pomijane	pomijane
BS_HATCHED	kolor linii wzoru	wzór wypełnienia
BS_PATTERN	pomijane	uchwyt mapy bitowej

Kreślenie na ekranie :

`COLORREF` **`SetPixel`**`( HDC hdc, int X, int Y, COLORREF crColor)` – ustawia piksel o określonym położeniu na dany kolor
`COLORREF` **`GetPixel`**`( HDC hdc, int X, int Y)` – zwraca kolor piksela o określonym położeniu.

Część funkcji nie określa miejsca od którego rozpoczyna kreślenia. Korzysta z bieżącej pozycji pióra. Można zmienić tą pozycję za pomocą:
`BOOL` **`MoveToEx`**`(HDC hdc, int X, int Y, LPPOINT lpPoint);`
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.

`BOOL` **`Polyline`**`(HDC hdc, CONST POINT *lppt, int cPoint)` – funkcja kreśli linie łączące kolejne punkty z tablicy punktów. Funkcja nie wykorzystuje, ani nie zmienia bieżącej pozycji pióra.
cPoint – określa ilość punktów w tablicy musi być większy od 1.

`BOOL` **`PolylineTo`**`(HDC hdc, CONST POINT *lppt, int cPoint)` - funkcja kreśli linie łączące kolejne punkty z tablicy punktów. Przy czym pierwszy punkt odpowiada bieżącej pozycji pióra; funkcja ustawia bieżącą pozycję w jej punkcie końcowym.

`BOOL` **`Rectangle`**`(HDC hdc, int xLeft, int yTop, int xRight, int yBottom)` – kreśli prostokąt za pomocą bieżącego pióra i wypełnia go za pomocą bieżącego pędzla.

`BOOL` **`Ellipse`**`(HDC hdc, int xLeft, int yTop, int xRight, int yBottom)` – kreśli elipsę za pomocą bieżącego pióra i wypełnia ją za pomocą bieżącego pędzla. Środek elipsy odpowiada środkowi podanego prostokąta.

`BOOL` **`RoundRect`**`(HDC hdc, int xLeft, int yTop, int xRight, int yBottom, int xCornerEllipse, int yCornerEllipse)` – kreśli prostokąt z zaokrąglonymi wierzchołkami za pomocą bieżącego pióra i wypełnia go za pomocą bieżącego pędzla. x`CornerEllipse i yCornerEllipse` określają szerokość i wysokość elipsy użytej do zaokrąglania prostokąta.

`BOOL` **`Arc`**`(HDC hdc, int xLeft, int yTop, int xRight, int yBottom, int xStart, int yStart, int xEnd, int yEnd)` – kreśli łuk, będący częścią elipsy opisanej przez 4 pierwsze parametry prostokąta. Pozostałe 4 określają początek i koniec łuku będący przecięciem elipsy i prostej łączącej środek elipsy i punkt początku / końca.

BOOL **Chord**(HDC hdc, int xLeft, int yTop, int xRight,
int yBottom, int xStart, int yStart, int xEnd, int yEnd)
– kreśli łuk jak w funkcji Arc, oraz odcinek łączącą koniec i początek łuku przy pomocy bieżącego pióra i wypełnia obszar pomiędzy łukiem a odcinkiem za pomocą bieżącego pędzla.

BOOL **Pie**(HDC hdc, int xLeft, int yTop, int xRight,
int yBottom, int xStart, int yStart, int xEnd, int yEnd)
– kreśli łuk jak w funkcji Arc, oraz odcinki łączącą koniec i początek łuku z środkiem elipsy przy pomocy bieżącego pióra i wypełnia obszar pomiędzy łukiem a odcinkami za pomocą bieżącego pędzla.

BOOL **PolyBezier**(HDC hdc, Const POINT *lppt, DWORD cPoints)– kreśli jedną lub więcej krzywych Beziera.
lppt – wskaźnik na tablicę punktów
Pierwsza krzywa jest kreślona od pierwszego do czwartego punktu, przy użyciu 2 i 3 punktu jako punktów kontrolnych (które pełnią rolę magnesów odciągających linię od odcinka łączącego punkty końcowe). Każda kolejny segment krzywej potrzebuje trzech punktów. Punkt końcowy wcześniejszego segmentu jest punktem początkowym następnego.
CPoints – ilość punktów w tablicy (3*ilość krzywych +1)

BOOL **PolyBezierTo**(HDC hdc, Const POINT *lppt, DWORD cPoints)– kreśli jedną lub więcej krzywych Beziera. Różni się od PolyBezier tym, że pierwszy punkt początkowy odpowiada bieżącej pozycji pióra. Funkcja ustawia bieżącą pozycję w jej punkcie końcowym.

BOOL **Polygon**(HDC hdc, CONST POINT *lppt, int cPoint) – funkcja kreśli linie łączące kolejne punkty z tablicy punktów. Jeżeli ostatni punkt z tablicy nie pokrywa się z pierwszym Windows dodaje odcinek zamykający figurę. Następnie system wypełnia figurę korzystając z pędzla bieżącego w sposób zależny od ustawionego trybu wypełniania wielokątów.
cPoint – określa ilość punktów w tablicy; musi być większy od 1.

int **SetPolyFillMode**(HDC hdc, int iMode) – zmienia tryb wypełniania wielokątów w bieżącym DC.
iMode – tryb wypełniania wielokątów:
ALTRNATE – domyślny. Obszar zostanie wypełniony jeśli półprosta wyprowadzona z dowolnego punktu obszaru przetnie nieparzystą liczbę krawędzi. Pozostałe obszary pozostaną niewypełnione.
WINDING – Windows wypełni wszystkie domknięte obszary.

int **FillRect**(HDC hdc, CONST RECT * rect, HBRUSH hBrush) –
wypełnia obszar prostokąta wskazany
przez rect za pomocą podanego pędzla. Zawiera lewą i górną krawędź prostokąta, nie zawiera krawędzi prawej i dolnej.
int **FrameRect**(HDC hdc, CONST RECT * rect, HBRUSH hBrush) – maluje pędzlem prostokątną ramkę,
środek pozostawiając nie wypełniony.
BOOL **InvertRect**(HDC hdc, CONST RECT * rect) – odwraca barwę pikseli wewnątrz prostokąta.

Operacje na obiekcie typu RECT:

`BOOL` **`SetRect`**`(RECT *rect, int xLeft, int yTop, int xRight, yBottom)` – funkcja ustawia pola
obiektu typu RECT.

`BOOL` **`OffsetRect`**`(RECT *rect,int x,int y)` –przesuwa prostokąt o określoną liczbę jednostek wzdłuż osi x i y.

`BOOL` **`InflateRect`**`(RECT *rect, int dx, int dy)` – zwiększa lub zmniejsza rozmiary prostokąta o określoną
liczbę jednostek w obie strony.

`BOOL` **`SetRectEmpty`**`(RECT *rect)` – wyzerowuje wszystkie pola zmiennej typu RECT.

`BOOL` **`CopyRect`**`(RECT *DestRect, RECT *SrcRect)` – Kopiuje pola jednego prostokąta do drugiego.

`BOOL` **`IntersectRect`**`(RECT *DestRect, RECT *SrcRect1, RECT *SrcRect2)` – wylicza część wspólną
dwóch prostokątów.

`BOOL` **`UnionRect`**`(RECT *DestRect,RECT *SrcRect1,RECT *SrcRect2)`- wylicza sumę dwóch prostokątów.

`BOOL` **`IsRectEmpty`**`(RECT *rect)` – zwraca TRUE jeżeli wszystkie boki prostokąta są zerowe.

`BOOL` **`PtInRect`**`(RECT *rect, POINT point)` – zwraca TRUE jeżeli punkt znajduje się wewnątrz prostokąta.

Region.

Region opisuje obszar powstały ze złożenia prostokątów, wielokątów i elips. Region podobnie jak pióro lub pędzel jest obiektem GDI. To znaczy można go utworzyć, pobrać do niego uchwyt, a następnie należy go usunąć (*DeleteObject(hRgn)*).

`HRGN` **`CreateRectRgn`**`(int xLeft, int yTop, int xRight, int yBottom)` – tworzy region w kształcie prostokąta.

`HRGN` **`CreateRectRgnIndirect`**`(CONST RECT * rect)` – tworzy region w kształcie prostokąta.

`HRGN` **`CreateEllipticRgn`**`(int xLeft, int yTop, int xRight, int yBottom)` – tworzy region w kształcie elipsy.

`HRGN` **`CreateEllipticRgnIndirect`**`(CONST RECT * rect)` – tworzy region w kształcie elipsy.

`HRGN` **`CreateRectRgn`**`(int xLeft, int yTop, int xRight, int yBottom, int nWidthEllipse, int nHeightEllipse)` – tworzy region w kształcie prostokąta z zaokrąglonymi rogami.

`HRGN` **`CreatePolygonRgn`**`(CONST POINT *lppt, int cPoint, int iPolyFillMode)` – funkcja tworzy region w kształcie wielokąta, powstałego w wyniku połączenia kolejnych punktów z tablicy punktów. Jeżeli ostatni punkt z tablicy nie pokrywa się z pierwszym Windows dodaje odcinek zamykający figurę.
cPoint – określa ilość punktów w tablicy; musi być większy od 1.
iPolyFillMode – jest równy ALTRNATE lub WINDING i określa które piksele należą do regionu.

`int` **`CombineRgn`**`(HRGN hDestRgn, HRGN hSrcRgn1, HRGN hSrcRgn2, int iCombine)` – funkcja składa dwa regiony dając w wyniku uchwyt nowego regionu. Wszystkie trzy regiony muszą istnieć wcześniej, a region poprzednio opisywany przez hDestRgn zostanie usunięty.
iCombine – określa sposób łączenia regionów
RGN_AND – część wspólna regionów
RGN_OR – suma regionów
RGN_XOR – suma regionów z wyłączeniem części wspólnej
RGN_DIFF – część należąca do hSrcRgn1 i nie należąca do hSrcRgn2
RGN_COPY – całość hSrcRgn1.
Funkcja zwraca:
NULLREGION – region pusty
SIMPLEREGION – pojedynczy prostokąt, wielokąt lub elipsa
COMPLEXREGION – złożenie prostokątów, wielokątów i elips.
ERROR - błąd
`BOOL` **`FillRgn`**`(HDC hdc,HRGN hrgn,HBRUSH hBrush)` – wypełnia obszar regionu za pomocą podanego pędzla.
`BOOL` **`FrameRect`**`(HDC hdc, HRGN hrgn, HBRUSH hBrush, int nWidth, int nHeight)` – maluje pędzlem ramkę dookoła regionu, środek pozostawiając nie wypełniony.
nWidth, nHeight – szerokość i wysokość ramki.
`BOOL` **`InvertRgn`**`(HDC hdc, HRGN hrgn)` – odwraca barwę pikseli wewnątrz regionu.
`BOOL` **`PaintRgn`**`(HDC hdc, HRGN hrgn)` – wypełnia region za pomocą bieżącego pędzla.
Regiony możemy wykorzystać do ustawiania obszarów w którym można malować – regionów obcinania. Poza tymi obszarami Windows nic nie wykreśli. Wykorzystujemy do tego funkcje
unieważniającą obszar: **`InvalidateRgn`**`(HWND hWnd , HRGN hRgn, BOOL bErase )` i zatwierdzającą obszar **`ValidateRgn`**`(HWND hWnd , HRGN hRgn)`.
wyznaczającą obszar obcinania dla danego DC:

```
HRGN SelectObject(HDC hdc, HRGN hRgn);
int SelectClipRgn(HDC hdc, HRGN hRgn);
```

Ścieżka.

Ścieżka jest zbiorem odcinków i łuków pamiętanych przez GDI. Podścieżki składają się z ciągu połączonych linii. Każda podścieżka może być zamknięta lub otwarta. Możemy zamknąć podścieżkę odcinkiem za pomocą funkcji:
`BOOL` **`CloseFigure`**`(HDC hdc)`
Ścieżkę tworzymy za pomocą funkcji:

```
BeginPath(hdc);
// funkcje kreślące linie, odcinki, krzywe
EndPath(hdc);
```

Funkcje usuwające definicję ścieżki:
`BOOL` **`StrokePath`**`(HDC hdc)` – kreśli ścieżkę za pomocą bieżącego pióra.
`BOOL` **`FillPath`**`(HDC hdc)` – zamyka odcinkiem każdą otwartą figurę w ścieżce i wypełnia ją używając pędzla bieżącego.

`BOOL` **`StrokeAndFillPath`**`(HDC hdc)` – kreśli ścieżkę za pomocą bieżącego pióra, zamyka odcinkiem każdą otwartą figurę w ścieżce i wypełnia ją używając pędzla bieżącego.
`HRGN` **`PathToRegion`**`(HDC hdc)` – przekształca ścieżkę w region
`BOOL` **`SelectClipPath`**`(HDC hdc, int iCombine)` – definiuje obszar obcinania. iCombine – przyjmuje takie same wartości jak *iCombine* w funkcji *CombineRgn* i określa sposób w jaki ścieżka zostanie połączona z bieżącym regionem obcinania.

Obrazy.

Wyróżniamy dwa sposoby przechowywania obrazów w programach: za pomocą map bitowych (cyfrowa reprezentacja obrazu) oraz metaplików (opisów obrazu).
Liczba kolorów mapy bitowej jest równa 2 do potęgi równej liczbie bitów przypadających na piksel (głębia koloru).(np. 16-kolorowa bitmapa– 4 bity na piksel).

DDB.

Do czasu Windows 3.0 mieliśmy do czynienia jedynie z bitmapami jako obiektami GDI. Są to mapy monochromatyczne, albo mapy o takiej samej organizacji kolorów jak rzeczywiste graficzne urządzenia wyjściowe (monitor, drukarka), dla którego mapa była tworzona. Działanie na tych bitmapach jest jednak wygodniejsze i wydajniejsze niż na bitmapach niezależnych sprzętowo, dlatego jeżeli potrzebujesz mapy bitowej zgodnej z kartą graficzną i wyłącznie na użytek twojego programu lepiej korzystać ze starszej wersji bitmapy.

Funkcje tworzące mapy bitowe zależne od sprzętu (DDB):

```
HBITMAP CreateBitmap(int cxWidth, int cyHeight, UINT iPlanes,
                     UINT iBitsPixel, CONST VOID * lpBits)
```

Pierwsze dwa argumenty odpowiadają szerokości i wysokości bitmapy w pikselach, trzeci określa liczbę warstw barwnych, a czwarty głębię koloru. W rzeczywistości trzeci i czwarty argument będą przyjmować następujące wartości:

- iBitPixel = iPlanes =1 – bitmapa monochromatyczna
- iBitPixel i iPlanes odpowiadają wartościom danego DC, które można pobrać przy pomocy *GetDeviceCap*.

lpBits- wskaźnik na tablicę kolorów, użytych do utworzenia bitmapy. Jeżeli równe NULL tworzymy nie zainicjowaną mapę bitową.

```
HBITMAP CreateCompatibeBitmap(HDC hdc, int cxWidth,
                              int cyHeight)
```

– na podstawie hdc Windows pobierze potrzebne informacje na temat warstw i głębi koloru. W tym przypadku mapa bitowa nie będzie zainicjowana. Oprócz posiadania takiej samej organizacji barw jak DC bitmapa nie jest związana w inny sposób z DC.

```
HBITMAP CreateBitmapIndirect(CONST BITMAP * bitmap)
```

– tworzy bitmapę w oparciu o strukturę **BITMAP**.

```
typedef struct tagBITMAP {
  LONG bmType;    // Ustawione na 0
  LONG bmWidth;   // Szerokość mapy bitowej w pikselach
  LONG bmHeight;  // Wysokość mapy bitowej wyrażona w wierszach pikseli
  LONG bmWidthBytes;       // Szerokość mapy bitowej wyrażona w bajtach – najmniejsza
parzysta liczba bajtów potrzebna do
                    // zapamiętania jednego wiersza pikseli.
  WORD bmPlanes; // Liczba warstw barwnych
  WORD bmBitsPixel;          // głębia koloru
  LPVOID bmBits;  // Wskaźnik do tablicy bitów
} BITMAP, *PBITMAP;
```

Aby skopiować zawartość tablicy zawierającej bity mapy bitowej do istniejącej mapy bitowej należy użyć funkcji:
`LONG SetBitmapBits(HBITMAP hBitmap, LONG dwCount, LPVOID pBitmap)`
dwCount - ilość bitów do przekopiowania do tablicy wskazanej przez pBitmap. Liczbę wszystkich bitów w mapie bitowej można wyliczyć za pomocą wzoru:
`dwCount = (DWORD) bitmap.bmWidhtBytes * bitmap.bmHeight * bitmap.bmPlanes;`

Możemy pobrać informacje o bitmapie za pomocą funkcji:
`GetObject(hBitmap, sizeof(BITMAP), (LPVOID) &bitmap)` – funkcja kopiuje informacje do struktury BITMAP pozostawiając pole bmBits puste.

Aby otrzymać dostęp do mapy bitowej należy wywołać funkcję:
`LONG GetBitmapBits(HBITMAP hBitmap, LONG dwCount, LPVOID pBitmap)`

W kontekście zwykłego urządzenia nie można wybrać mapy bitowej. Jest to możliwe jedynie w kontekście urządzenia pamięciowego. Kontekst urządzenia pamięciowego ta taki DC, który ma „powierzchnie wyświetlania" istniejącą tylko w pamięci komputera i możemy go utworzyć za pomocą funkcji:
`HDC CreateCompatibleDC(HDC hdc)` – funkcja tworzy DC pamięciowego , który ma "powierzchnię wyświetlania" zawierającą dokładnie 1 piksel monochromatyczny.
Aby powiększyć tak utworzony DC pamięciowego należy wybrać mapę bitową w DC:
`SelectObject(hdcMem, hBitmap);`
Z tak utworzonym DC możesz robić to samo co z zwykłym DC – zmieniać atrybuty, sprawdzać ich wartości, wybierać w kontekście pióro, pędzel i regiony.
Wszelkie zmiany poczynione na mapie bitowej pojawiają się na powierzchni wyświetlania DC pamięciowego. Wszystko co narysujesz w DC pamięciowego jest tak naprawdę rysowane na mapie bitowej.

Bitmapę będziemy wyświetlać w DC obszaru roboczego naszego okna przekazując bity bitmapy między kontekstami urządzenia.

`BOOL PatBlt(HDC hdc, int xDest, int yDest, int xWidth, int yHeight, DWORD dwROP)` – funkcja rysuje prostokąt używając bieżącego pędzla. Funkcja dokonuje operacji logicznych na kolorach pędzla i ekranu.
xDst, yDst – punkt logiczny wskazujący na lewy górny narożnik prostokąta dla trybu MM_TEXT. W pozostałych trybach narożnik prostokąta zależ od znaku przy dwóch kolejnych parametrach.
xWidth, yHeight – wartość bezwzględna z tych wartości wyznacza szerokość i wysokość prostokąta
dwROP – określa operacje wykonywane na kolorach pędzla i ekranu:
BLACKNESS – wypełnia prostokąt czarnym kolorem
WHITENESS – wypełnia prostokąt białym kolorem
DSTINVERT – odwraca kolor ekranu
PATINVERT – łączy kolory za pomocą operacji logicznej XOR (Pędzel ^ Ekran)
PATCOPY – przekopiowuje kolory pędzla

`BOOL` **`BitBlt`**`(HDC hdcDest, int xDest, int yDest, int xWidth, int yHeight, HDC hdcSrc, int xSrc, int ySrc, DWORD dwROP)` – funkcja przesyła bity kolorów określonego prostokąta z DC źródłowego do
DC docelowego. DC źródłowy może równać się DC docelowy
dwROP – określa operacje wykonywane na kolorach pędzla hdcDest, ekranu i prostokąta źródłowego.
BLACKNESS – wypełnia prostokąt czarnym kolorem
WHITENESS – wypełnia prostokąt białym kolorem
DSTINVERT – odwraca kolor ekranu
MERGECOPY – łączy kolor pędzla hdcDest z kolorem prostokąta źródłowego przy pomocy operacji logicznej AND.
MERGEPAINT – łączy odwrócone kolory prostokąta źródłowego i kolory ekranu przy pomocy operacji logicznej OR.
SRCCOPY – przekopiowuje kolory prostokąta źródłowego
NOTSRCCOPY – przekopiowuje odwrócone kolory prostokąta źródłowego.
SRCERASE - łączy odwrócone kolory ekranu i kolory prostokąta źródłowego przy pomocy operacji logicznej AND
NOTSRCERASE - łączy kolory ekranu i prostokąta źródłowego przy pomocy operacji logicznej OR i odwraca wynik.
PATINVERT – łączy kolory pędzla i ekranu za pomocą operacji logicznej XOR (Pędzel ^ Ekran)
PATCOPY – przekopiowuje kolory pędzla
PATPAINT – łączy kolory pędzla i odwrócone kolory prostokąta źródłowego przy pomocy operacji OR. Rezultat łączy z kolorami ekranu przy pomocy operacji OR.
SRCAND – łączy kolory prostokąta źródłowego i ekranu za pomocą operacji AND.
SRCINVERT - łączy kolory prostokąta źródłowego i ekranu za pomocą operacji XOR.
SRCPAINT - łączy kolory prostokąta źródłowego i ekranu za pomocą operacji OR.

`BOOL` **`StretchBlt`**`(HDC hdcDest, int xDest, int yDest, int xDestWidth, int yDestHeight, HDC hdcSrc, int xSrc, int ySrc, int xSrcWidth, int ySrcHeight, DWORD dwROP)` – funkcja rozszerza możliwości BitBlt o zmianę wielkości bitmapy z kontekstu urządzenia docelowego. Umośliwia także odwrócenie obrazu : odbicie zwierciadlane (jeżeli xSrcWidth i xDestWidth po zmianie na współrzędne urządzenia mają różne znaki) lub w kierunku góra-dół (jeżeli ySrcHeight i yDestHeight po zmianie na współrzędne urządzenia mają różne znaki). To w jaki sposób funkcja zmniejsza obraz zależy od trybu rozciągania mapy bitowej *(stretching mode)* określanego dla każdego DC.
Można zmieniać wartość tego trybu za pomocą funkcji:
`int` **`SetStretchBltMode`**`(HDC hdc, int iMode)`
iMode:
BLACKONWHITE = STRETCH_ANDSCANS – jeżeli dwa lub więcej pikseli trzeba złączyć w jeden piksel *StretchBlt* wykonuje na pikselach operację koniunkcji (logiczne AND)
WHITEONBLACK = STRETCH_ORSCANS - *StretchBlt* wykonuje na pikselach operację alternatywy(logiczne OR)
COLORONCOLOR = STRETCH_DELETESCANS – *StretchBlt* eliminuje wiersze lub kolumny bez wykonywania jakichkolwiek operacji logicznych.
HALFTONE = STRETCH_HALFTONE – Windows oblicza średni kolor w oparciu o piksele które trzeba połączyć.

DIB.
W Windows 3.0 zdefiniowany został nowy format map bitowych DIB (device-independent bitmap) – mapy bitowe niezależne od sprzętu. DIB posiada własną tabelę barw która określa metodę przekodowania bitów piksela na kolory systemu RGB. Format DIB służy głównie do wymiany obrazów między programami. Można go zapisywać w pliku albo skopiować do schowka. DIB nie jest obiektem GDI.

Metapliki.
Metapliki są binarnie zakodowanym zbiorem wywołań funkcji graficznych.

Najpierw tworzymy kontekst urządzenia metapliku za pomocą funkcji:
`HDC CreateMetaFile( LPCTSTR lpszFile)`
lpszFile – nazwa pliku (najczęściej z rozszerzeniem WMF (*windows metafile*)) lub NULL. Jeżeli NULL metaplik powstanie w pamięci.
Następnie kreślimy w DC metapliku. Wszystkie wywołania funkcji GDI są zapisywane binarnie w metapliku.
Kolejny etap to zamknięcie kontekstu urządzenia metapliku za pomocą funkcji:
`HMETAFILE CloseMetaFile(HDC hdcMeta)` – funkcja zwraca uchwyt metapliku.
Następnie możemy wyświetlić obraz zapamiętany w metapliku w DC za pomocą funkcji:
`BOOL PlayMetaFile(HDC hdc, HMETAFILE hmf)`
Podobnie jak inne obiekty GDI, metapliki powinny zostać usunięte przed zakończeniem programu. W tym celu wywołujemy funkcje:
`BOOL DeleteMetaFile(HMETAFILE hmf)`

Z tworzonym w ten sposób plikami wiążą się pewne problemy. Na przykład program używający metapliku utworzonego w innym programie nie może w prosty sposób określić wymiarów obrazu. Dlatego metapliki są rzadko wykorzystywane do wymiany rysunków między programami.
W związku z typ powstał nowy format: „ulepszone metapliki” (enhanced metafile – EMF)
Tworzenie „ulepszonych metaplików” i ich wyświetlanie przebiega podobnie jak starych metaplików.
`HDC CreateEnhMetaFile(HDC hdc, LPCTSTR lpszFile, CONST RECT * lpRect, LPCTSTR lpDescription)`
hdc – Windows wykorzystuje ten parametr do wstawienia do nagłówka metapliku informacji o wymiarach rysunku. Jeżeli =NULL GDI pobierze odpowiednie informacje z kontekstu urządzenia wyświetlającego.
lpRect – wskazuje na prostokąt wyznaczający całkowite wymiary metapliku. Jeżeli =NULL to GDI określi te wymiary.
lpDescription – tekst opisujący metaplik. Składa się z dwóch części. Pierwsza zakończona 0 zawiera nazwę aplikacji, druga zakończona dwoma zerami opisuje rysunek. Może równać się NULL.
`HENHMETAFILE CloseEnhMetaFile(HDC hdcEMF)` – funkcja usuwa uchwyt DC metapliku otrzymując w zamian uchwyt metapliku.
`BOOL PlayEnhMetaFile(HDC hdc, HENHMETAFILE hmf, CONST RECT * lpRect)`
lpRect – prostokąt do którego GDI dopasowuje rysunek.
`BOOL DeleteEnhMetaFile(HENHMETAFILE hmf)` – usuwa metaplik z pamięci
`HENHMETAFILE GetEnhMetaFile(LPCTSTR lpszFile)` – funkcja zwraca uchwyt do metapliku na dysku.

`UINT` **`GetEnhMetaFileHeader`**`(HENHMETAFILE hemf, UINT cbSize, LPENHMETAHEADER emh)` – funkcja zwraca informacje z nagłówka metapliku.
cbSize – określa w wielkość bufora na dane.
emh – wskaźnik na strukturę ENHMETAHEADER, jeżeli =NULL funkcja zwraca wielkość rekordu nagłówkowego
Użyteczne pola **ENHMETAHEADER**:
RECT rclBounds – zawiera wymiary rysunku w pikselach
RECT rclFrame – zawiera wymiary rysunku w setnych częściach milimetra.

`UINT` **`GetEnhMetaFileDescription`**`(HENHMETAFILE hemf, UINT sizechBuffer, LPTSTR lpszDescription)` – wypełnia bufor opisem metapliku. Jeżeli lpszDescription=NULL funkcja zwraca długość opisu.

Teksty i czcionki

Funkcja wyświetlającą tekst:
`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` – liczb znaków w ciągu.
Funkcja określająca położenie napisów, zmieniająca domyślne znaczenie parametrów *`nXStart`*, *`nYStart`* z funkcji *TextOut.* :
`UINT` **`SetTextAlign`**`(HDC hdc, UINT fmode)`
fmode:
TA_LEFT – domyślny; wyrównuje tekst do lewej krawędzi. nXStart wskazuje na punkt od którego rozpoczynamy pisanie.
TA_RIGHT – łańcuch znaków dosunięty prawą krawędzią ostatniego znaku do punktu o współrzędnej poziomej nXStart.
TA_CENTER – współrzędna nXStart wyznacza środek kreślonego napisu
TA_TOP – domyślny; nYStart wyznacza górną krawędź znaków
TA_BOTTOM – wszystkie znaki ciągu znajdują się powyżej rzędnej nYStart
TA_BASELINE – podstawa wyświetlania tekstu (linia która biegnie wzdłuż podstawy liter nie mających tzw. wydłużeń dolnych np. (wydłużenia mają np.: y,p,g)) znajdzie się na rzędnej nYStart
TA_UPDATECP – system ignoruje nXStart i nYStart, a zamiast nich użyje bieżącej pozycji pióra. Dla tego stylu TextOut
ustawi bieżącą pozycję pióra na koniec wykreślonego teksu (dla TA_LEFT) lub na jego początku (dla TA_RIGHT)
`BOOL` **`SetTextJustification`**`(HDC hdc, int nExtra, nBrakCount);-`
ustawia wielkość przestrzeni jaką trzeba rozłożyć między przerwy w tekście.
nExtra – określa przestrzeń jaką trzeba rozłożyć pomiędzy przerwy w łańcuchach znaków.
NBreakCount – liczba przerw w łańcuch, do których należy dodać wolną przestrzeń.
Zaczynając nowy wiersz jeżeli wcześniej użyto tej funkcji należy ustawić przerwy:
`SetTextJustification(hdc, 0,0);`

Funkcja wyświetla tekst zamieniając umieszczone w tekście znaki tabulacji (‘\t’ lub 0x09) na odstępy określone w tablicy liczb całkowitych:

```
LONG TabbedTextOut(HDC hdc, int nXStart, int nYStart, LPCTSTR
lpString, int cbString, int iNumTab, CONST LPINT piTabStops,
int xTabOrgin)
```

iNumTab – liczba znaków tabulacji
piTabStops – tablica współrzędnych tabulacji wyrażonych w pikselach
Jeżeli iNumTab=0 i piTabStops=NULL system ustawia tabulatory co 8 znak (w oparciu o średnią szerokość znaku)
Jeżeli iNumTab=1 to piTabStops zawiera liczbę określającą odstęp pomiędzy kolejnymi tabulatorami.
xTabOrgin – wskazuje na punkt na os x od którego nastąpi odmierzanie tabulatorów.
Jeżeli funkcja zakończy się sukcesem zwraca rozmiar napisu HIWORD(wynik) – wysokość, LOWORD(wynik) - szerokość

```
BOOL ExtTextOut(HDC hdc, int nXStart, int nYStart, UINT
iOptions, CONST RECT *rect, LPCTSTR lpString, UINT cbString,
CONST INT *lpDx)
```

iOptions – określają w jaki sposób interpretować prostokąt rect. Może być równe 0.
ETO_CLIPPED – rect jest to prostokąt obcinania
ETO_OPAQUE – rect – prostokątne tło, które zostanie wypełnione kolorem tła bieżącego.
lpDx – wskazuje na tablicę liczb całkowitych określających odstępy pomiędzy kolejnymi znakami w ciągu. Jeżeli ten parametr równa się NULL domyślny odstęp zostanie zastosowany..

Funkcja wyświetlająca sformatowany tekst w podanym prostokącie:

```
BOOL DrawText(HDC hdc, LPCTSTR lpString, int cbString, LPRECT
lpRect, UINT iFormat)
```

Jeżeli cbString =-1 i lpString jest ciągiem zakończonym 0 to Windows sam obliczy długość znaku.
iFormat – określa sposób formatowania tekstu:
0 – system interpretuje tekst jako zbiór wierszy oddzielonych znakiem powrotu karetki (‘\r’ lub 0x0D) lub znakiem następnej linii (‘\n’ lub 0x0A).
DT_LEFT – domyślny. Wyrównanie lewostronne tekstu
DT_RIGHT – wyrównanie prawostronne
DT_CENTER – wyśrodkowuje tekst pomiędzy lewym i prawym bokiem prostokąta
DT_SINGLELINE – nie interpretuje znaków powrotu karetki oraz przejścia do następnej linii jako znaki końca wiersza
DT_TOP – domyślny; wyrównuje tekst do górnej części prostokąta.
DT_BOTTOM – wyrównanie do dolnej części prostokąta
DT_VCENTER – wyrównanie tekstu pomiędzy dolnym i górnym bokiem prostokąta
DT_WORDBREAK – wymusza łamanie wiersza po dojściu do krawędzi prostokąta.
DT_NOCLIP – kreśli tekst bez obcinania fragmentów nie mieszczących się w prostokącie
DT_EXTERNALLEADING – odstępy między wierszami wysokości znaku bez pominięcia dodatkowego odstępu zalecanego przez projektanta czcionki.
DT_EXPANDTABS - interpretuje znaki tabulacji. Domyślnie znaki tabulacji występują w odstępach równych ośmiokrotnej średniej szerokości znaku.

Napisy są wyświetlane dla danego kontekstu urządzenia. Kilka atrybutów DC dotyczy napisów.
Kolor liter domyślnie jest czarny zmieniamy go za pomocą funkcji:
`COLORREF` **`SetTextColor`**`(HDC hdc, COLORREF rgbColor)` – funkcja zwraca poprzedni kolor tekstu.
`COLORREF` **`GetTextColor`**`(HDC hdc)` – funkcja zwraca bieżący kolor napisów.
Odstęp między znakami jest wypełniony kolorem uzależnionym od trybu tła oraz koloru tła.
Można korzystać z kolorów systemowych pobieranych przy pomocy funkcji:
GetSysColor(int nIndex)
m.in nIndex może być równe:
COLOR_GRAYTEXT – szary (nieaktywny) kolor
COLOR_HIGHLIGHT – kolor używany do zaznaczania wybranego elementu w kontrolce
COLOR_HIGHLIGHTTEXT – kolor tekstu używany do zaznaczania wybranego elementu w kontrolce
COLOR_WINDOW – tło okna
COLOR_WINDOWTEXT – kolor tekstu w oknie

Czcionki.

Można pobrać predefiniowane czcionki systemowe:
```
HFONT hFont = GetStockObject(iFont);
```
iFont równa się SYSTEM_FONT (domyślna czcionka – o proporcjonalnej szerokości liter) lub SYSTEM_FIXED_FONT(o takiej samej szerokości znaków)
```
SelectObject(hFont);
```

Czcionki GDI znajdujące się w plikach na dysku twardym dzielą się na trzy kategorie:
Rastrowe – każdy znak jest zakodowany w postaci wzorca pikseli; „nieskalowalnymi" bez pogorszenia jakości; wydajność; dobra czytelność.
Wektorowe – znak w postaci zbioru połączonych odcinków. – niska wydajność wyświetlania, czytelność słaba przy małych wymiarach; przy dużych brzydkie; Skalowalne.
TrueType – zdefiniowane przez kontur składający się z odcinków i łuków. Kontur jest wykorzystywany do stworzenia mapy bitowej znaku.
Windows 95 był wyposażony w 13 czcionek TrueType w kolejnych wersjach Windows lista czcionek uległa powiększeniu.
(Courier New , **Courier New Bold**, *Courier New Italic*, ***Courier New Bold Italic***, Times New Roman, **Times New Roman Bold**, *Times New Roman Italic*, ***Times New Roman Bold Italic***, Arial, **Arial Bold**, *Arial Italic*, ***Arial Bold Italic***, Symbol).

```
HFONT CrateFontIndirecta(LPLOGFONT lplf)
typedef struct tagLOGFONT {
  LONG lfHight     // pożądana wysokość czcionki w jednostkach logicznych. 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          // kursywa jeżeli !=0
  BYTE lfUnderline
  BYTE lfStrikeOut
  BYTE lfCharSet
  BYTE lfOutPrecision
  BYTE lfClipPrecision
  BYTE  lfQuality
  BYTE lfPitchAndFamily
  TCHAR lfFaceName
} LOGFONT, *PLOGFONT
```
Aby wybrać czcionkę TrueType większość wartości powinna być równa 0.

`BOOL GetTextExtentPoint32(HDC hdc, LPCTSTR lpString, int iCount, LPSIZE lpSize)` – funkcja podaje szerokość i wysokość łańcucha znaków na podstawie bieżącej czcionki w DC.
Struktura SIZE składa się z dwóch pól typu LONG cx, cy.