Урок 3. Рисование (vc7)

Рисование происходит на так называемом контексте устройства (Device context). Идея следующая: рисование на экране, на принтере и пр. – однотипно. Вы рисуете на некотором абстрактном DC. Если DC связать с окном на экране, то рисование будет в происходить в окне; если связать его с принтером – то на принтере; если с файлом – то, соответственно, в файл.

Если рисование происходит внутри функции OnDraw, то в качестве параметра функции передается указатель на уже созданный приложением DC, поэтому его можно просто использовать, не заботясь о создании и удалении.

Если рисование происходит в какой-то другой функции (как правило, из класса CNameView, в ответ на какое-нибудь событие, например, на выбор пункта меню), то этот контекст нужно получить. Это делается так:

CClientDC dc(this);//соотносим DC с данным окном

Можно заводить DC также динамически

CClientDC * pdc=new CClientDC(this);

но тогда надо не забыть потом его удалить.

Указатель на pdc можно сделать членом класса CNameView, тогда можно в одной из функций класса, начинающей рисование, создать CClientDC * pdc=new CClientDC(this);

а в другой, завершающей процесс рисования, – удалить delete pDC;

Контекст устройства является набором различных составляющих. В частности, туда входит карандаш для рисования (класс CPen) и кисть для закрашивания (CBrush). У них стандартные атрибуты, например, черный цвет у карандаша. Если вы хотите поменять, например, цвет, то нужно завести карандаш с таким цветом. Затем добавить его в DC взамен старого, а указатель на старый обязательно сохранить. После использования карандаша нужно старый карандаш вернуть на место. Вот пример изменения цвета.

```
void CMyView::OnDraw( CDC* pDC )
{
    CPen penBlack;  // Construct it, then initialize
    // CreatePen возвращает 0, если создать объект не удалось
    penBlack.CreatePen( PS_SOLID, 2, RGB(0,0,0) ) )
    // Select it into the device context
    // Save the old pen at the same time
    CPen* pOldPen = (CPen *) pDC->SelectObject( &penBlack );

    // Draw with the pen. For example,
    pDC->MoveTo(10,20);
    pDC->LineTo(100,200);

    // Restore the old pen to the device context
    pDC->SelectObject( pOldPen );
}
```

Цвет кодируется тремя числами: R(red)G(green)B(blue), каждое от 0 до 255. Т.о., (0,0,0) – черный, (255,255,255) – белый, (0,255,0) - зеленый. PutPixel(255,0,0); - рисование точки красного цвета. RGB(c1,c2,c3) – это макрос, в результате которого образуется число. Поэтому хранить цвет можно в виде одного числа.

Задание:

Сделать приложение, рисующее линию и закрашенный прямоугольник на экране.

Добавить пункты меню, которые меняют цвет на синий, красный или зеленый.

При изменении размеров окна цвет должен оставаться тем, который был выбран. Для этого цвет (или весь карандаш целиком) должен храниться в классе CNameView.

Дополнительные сведения

1. Если нужно один и тот же объект (например, карандаш или кисть) создавать несколько раз (меняя цвет, например), то последовательность действий такая:

```
CPen pen;
CPen * pOldPen;

for(int i=0; i<3; ++i)
{
      pen.CreatePen( PS_SOLID, i, RGB(i*100,0,0) );
      pOldPen = pDC->SelectObject( &pen );

      // Draw with the pen. For example,
      pDC->MoveTo(10*(i+1),20);
      pDC->LineTo(100*(i+1),200);
      // Restore the old pen to the device context
      pDC->SelectObject( pOldPen );
      // Delete the graphic object; действует только если перед этим
      // выбрать другой графический объект, тем самым освободив предыдущий.
      pen.DeleteObject();
}
```

2. Есть набор стандартных карандашей и кистей. Их можно использовать, не создавая своих объектов. Например, так:

```
CBrush* pOldBrush = (CBrush*) pDC->SelectStockObject(BLACK_BRUSH);
```

3. Есть возможность создать не саму кисть (или перо), а ее содержание (логическую структуру). Потом на основе этого содержания можно создавать уже настоящие кисти/перья. Логическая структура удобна тем, что с ней можно работать, как с настоящей структурой, менять ее свойства через изменение полей.

```
LOGBRUSH logBrush;
logBrush.lbStyle = BS_HATCHED;
logBrush.lbColor = RGB(192, 192, 0);
logBrush.lbHatch = HS_CROSS;
// Declare an uninitialized CBrush ...
CBrush brush;
// ... and initialize it with the LOGBRUSH.
brush.CreateBrushIndirect(&logBrush);
```

4. Чтобы рисовать в относительных координатах, можно узнать размер всего окна (его клиентской (внутренней) части).

```
   CRect rc; //Стандартный класс прямоугольника
   GetClientRect(&rc); //Получаем прямоугольник, имеющий размеры области для
рисования в координатах устройства
   pDC->Rectangle(0, 0, rc.Width() / 2, rc.Height() / 2); //Прямоугольник размером
//в четверть окна в логических координатах; в данном случае логические и
//координаты устройства совпадают, иначе надо пользоваться функциями DPtoLP и
LPtoDI
```

5. Со шрифтами работают так же, как и с перьями и кистями. Ниже приведен пример работы с использованием логической структуры.

```
CFont font;
LOGFONT lf;
memset(&lf, 0, sizeof(LOGFONT));       // zero out structure
lf.lfHeight = 14;                      // request a 12-pixel-height font
strcpy(lf.lfFaceName, "Arial");        // request a face name "Arial"
font.CreateFontIndirect(&lf);  // create the font
CFont* def_font = pDC->SelectObject(&font);
pDC->TextOut(5, 5, "Hello"); //по умолчанию указанные координаты – это координаты
// верхнего левого угла прямоугольника, в который идет вывод текста. Это можно //
// изменить с помощью вызова pDC->SetTextAlign(…)
pDC->SelectObject(def_font);
font.DeleteObject();// Done with the font. Delete the font object.

lf.lfHeight = 20;                      // request a 20-pixel-height font
strcpy(lf.lfFaceName, "Courier New");        // request a face name "Arial"
font.CreateFontIndirect(&lf);  // create the font
def_font = pDC->SelectObject(&font);
pDC->TextOut(50, 50, "Hello");
pDC->SelectObject(def_font);
```