

*Интерфейс графических
устройств
GDI*

Интерфейс графических устройств (Graphical Device Interface, GDI)

- GDI представляет собой совокупность программных средств Windows, организующих вывод на различные устройства вывода
 - на экран,
 - на устройства печати
 - в файлы
 - все многообразие графических объектов:
 - текстовых строк,
 - геометрических фигур,
 - растровых изображений
 - др.

GDI предоставляет программисту более двухсот функций для управления режимами вывода и построения на экране требуемых изображений.

- функции для **создания инструментов** рисования
 - цветные кисти и перья, шрифты различных гарнитур;
- функции **управления цветами**;
- функции **получения и задания режимов рисования**;
- функции **вывода тех или иных объектов** и т.д.

- Помимо самого вывода изображений, в задачу интерфейса GDI входит **наблюдение за границами**, в которых осуществляется этот вывод. Так, программа может попытаться вывести на экран очень длинную текстовую строку или другое изображение большого размера.
- Однако GDI отобразит на экране только те части этих объектов, которые попадают **внутри окна приложения**.
 - В результате приложения никогда не затирают друг друга, сосуществуя в пределах своих окон.

GDI участвует в организации графической оболочкой многослойного экранного кадра

- на экране может быть одновременно изображено несколько окон одного или различных приложений,
- окна верхнего уровня отображаются целиком, а от окон нижних уровней видны только выступающие части.
- Для того чтобы это было возможным, перерисовка каждого окна должна вестись только в пределах видимых в настоящий момент границ.
- Для успешного использования графических возможностей Windows в прикладных программах программист должен
 - не только знать состав и особенности применения многочисленных функций, связанных с изображением на экране графических объектов (это, кстати, наиболее простая задача),
 - но и понимать принципы динамического взаимодействия системы с приложением в процессе организации экранного кадра.

Одним из наиболее важных системных средств является сообщение WM_PAINT

//Программа Сообщение WM PAINT

*/*Операторы препроцессора*/*

#include <windows.h> //Два файла с определениями, макросами
#include <windowsx.h> //и прототипами функций Windows

*/*Прототипы используемых в программе функций
пользователя*/*

•

*LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); //Оконная
функция*

void OnPaint(HWND); //Прототип функции OnPaint

void OnDestroy(HWND); //Прототип функции OnDestroy

/ Главная функция WinMain*/*

```
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, int)  
{  
  char szClassName[]="MainWindow"; //Произвольное имя класса главного окна  
  char szTitle[]="Программа GDI_1"; //Произвольный заголовок окна  
  MSG Msg; //Структура Msg типа MSG для  
временного хранения сообщений Windows  
  WNDCLASS wc; //Структура wc типа WNDCLASS для  
задания характеристик окна
```


/ Регистрация класс главного окна*/*

```
memset (&wc, 0, sizeof (wc) );           //Обнуление всех членов структуры wc  
wc.lpszClassName=szClassName;          //Определяем оконную процедуру для главного окна  
wc.hInstance=hInst;                    //Дескриптор приложения  
wc.hIcon=LoadIcon(NULL,IDI_APPLICATION); //Стандартная иконка  
wc.hCursor=LoadCursor(NULL, IDC_ARROW); //Стандартный курсор мыши  
wc.hbrBackground=GetStockBrush(WHITE_BRUSH); //Белая кисть для фона окна  
wc.lpszClassName=szClassName;          //Класс главного окна
```



```
RegisterClass(&wc);                     //Вызов функции Windows регистрации класса окна
```


*/*Создадим главное окно и сделаем его видимым*/*

```
HWND hwnd=CreateWindow(szClassName,szTitle,//Класс и заголовок окна  
WS_OVERLAPPEDWINDOW,10,10,300,100,//Стиль окна, его координаты и размеры  
HWND_DESKTOP,NULL,hInst,NULL); //Родитель, меню, другие параметры  
  
ShowWindow(hwnd,SW_SHOWNORMAL); //Вызов функции Windows показа окна
```

*/*Организуем цикл обработки сообщений*/*

while(GetMessage(&Msg,NULL,0,0)) //Цикл обработки сообщений: ждать

DispatchMessage(&Msg); //сообщения, записать его в msg и передать WndProc

return 0; //После выхода из цикла обработки

//сообщений вернуться в Windows

} //Конец функции WinMain

*/*Оконная функция главного*

```
LRESULT CALLBACK WndProc(HWND hwnd,UINT msg,WPARAM wParam,LPARAM lParam)
{
  switch(msg)
  {
    //Переход по значению msg - коду сообщения
    HANDLE_MSG(hwnd,WM_PAINT,OnPaint); //При поступлении сообщения
    WM_PAINT
    HANDLE_MSG(hwnd,WM_DESTROY,OnDestroy); //При завершении пользователем
    default: //В случае всех остальных сообщений
    Windows обработка их
    return(DefWindowProc(hwnd,msg,wParam,lParam)); //по умолчанию
  }
  //Конец оператора switch
}
//Конец функции WndProc
```

- Ранее в ней обрабатывалось лишь одно сообщение WM_DESTROY. В настоящем примере прикладной обработке подвергается также сообщение WM_PAINT.
- Цель обработки - вывод на экран заданной текстовой строки.
- Каждый раз, когда в приложение поступает сообщение WM_PAINT, эта строка заново выводится в главное окно приложения.
- В результате при любых манипуляциях с окном приложения
 - сокращении
 - растягивании,
 - сворачивании в значок
 - разворачивании на весь экран

*/*Функция обработки сообщения WM_DESTROY*/*

```
void OnDestroy(HWND)  
{  
    PostQuitMessage(0); //Вызов функции Windows завершения приложения  
}
```

*/*Функция обработки сообщений WM_PAINT*/*

```
void OnPaint(HWND hwnd){  
    char szText[]="Строка текста для вывода в главное окно";  
    PAINTSTRUCT ps;          //Структура, требуемая для рисования в рабочей области  
    HDC hdc=BeginPaint(hwnd,&ps);    //Получение контекста устройства  
    TextOut(hdc,5,30,szText,strlen(szText));    //Вывод строки текста с точки 5,30  
    EndPaint(hwnd,&ps);          //Освобождение контекста устройства  
}          //Конец функции OnPaint()
```



Функция OnPaint()

Программный блок обработки сообщений WM_PAINT выделен в функцию OnPaint(). Соответственно в раздел прототипов включен прототип этой функции.

```
void OnPaint(HWND hwnd){  
    char szText[]="Строка текста для вывода в главное окно";  
    PAINTSTRUCT ps;          //Структура, требуемая для рисования в рабочей области  
    HDC hdc=BeginPaint(hwnd,&ps);    //Получение контекста устройства  
    TextOut(hdc,5,30,szText,strlen(szText));    //Вывод строки текста с точки 5,30  
    EndPaint(hwnd,&ps);          //Освобождение контекста устройства  
}          //Конец функции OnPaint()
```

Рассмотрим более детально роль сообщения WM_PAINT и процедуру его обработки.

Обработка сообщений WM_PAINT

- Сообщения WM_PAINT выделяются среди всех остальных тем, что их обработка включается практически в любое приложение Windows, если в нем хоть что-нибудь рисуется на экране.
- Общее правило рисования заключается в том, что вывод в окно приложения любых графических объектов
 - текстовых строк,
 - геометрических фигур,
 - отдельных точек,
 - растровых изображений -должен выполняться *исключительно в процедуре обработки сообщения WM_PAINT.*
- Только в этом случае графическое содержимое окна не будет теряться при загораживании данного окна окнами других приложений.
- Каким образом вообще Windows поддерживает содержимое своих многочисленных окон?

Выделим несколько типичных ситуаций.

- Если мы с помощью мыши или клавиатуры перемещаем окно приложения по экрану, то этот процесс не затрагивает само приложение.
 - Копирование содержимого окна по мере его перемещения на новые места экрана обеспечивают системные программы Windows.
- Если в приложении имеется линейка меню, то при разворачивании пунктов меню они перекрывают часть окна и, следовательно, при их сворачивании заслоненную ранее область окна надо перерисовать.
 - Эту задачу Windows также берет на себя, сохраняя в своей памяти заслоняемое изображение и выводя его на экран при сворачивании меню.
- По-другому обстоит дело, если перерисовка окна потребовалась в результате
 - разворачивания окна, свернутого ранее в пиктограмму,
 - при растягивании ранее сжатого окна или
 - при перемещении по пространству главного окна порожденного окна диалога.

В этих случаях, когда поврежденной может оказаться значительная часть окна или даже окно целиком (как при разворачивании пиктограммы), Windows уже не берет на себя задачу сохранения и восстановления изображения в окне, а вместо этого посылает в приложение сообщение WM_PAINT.

Сообщение WM_PAINT

- Программа в ответ на сообщение WM_PAINT должна сама восстановить все, что должно изображаться в окне.
- Windows сообщает в приложение, какая часть окна требует перерисовки,
- приложение в принципе может перерисовывать только поврежденную часть окна, что заметно сократило бы временные издержки на вывод изображения.
- Однако, такой алгоритм рисования составить слишком сложно, если вообще возможно, и в ответ на сообщение WM_PAINT программа вынуждена заново рисовать все, что должно изображаться в окне.

Рабочая область окна

- Главное окно приложения обычно имеет
 - заголовок с управляющими кнопками
 - толстую рамку
 - линейку меню.
- Все эти элементы окна образуют нерабочую область окна (nonclient area, неклиентская область) и обычно недоступны программе.
- Остальная часть окна, куда программа может выводить что угодно, называется рабочей областью (client area, область клиента)
- Восстановление нерабочей области при любых манипуляциях с окном Windows берет на себя, программа обязана восстанавливать только рабочую область.

Контекст устройства.

- Обработка сообщения WM_PAINT связана с использованием важнейшего поля данных Windows, называемого *контекстом устройства*.
- Контекст устройства представляет собой системную область памяти, закрепляемую за рабочей областью окна, в которой хранятся текущие значения режимов, связанных с рисованием, а также дескрипторы инструментов рисования - кисти, пера, шрифта и пр.
- Все графические функции GDI используют контекст устройства для определения режима рисования и характеристик применяемых ими инструментов.

Пример

- функция вывода линии получает из контекста устройства
 - толщину
 - цвет пера(через дескриптор пера) которым должна быть нарисована линия;
- функции вывода геометрических фигур, в добавление к характеристикам пера, получают
 - цвет и
 - фактуру кистидля закрашивания (заливки) рисуемых фигур;
- функции вывода текста получают (через дескриптор шрифта) все необходимые характеристики шрифта
 - гарнитуру,
 - размер,
 - цвет,
 - насыщенность (жирность) и пр.

Дескриптор контекста

- Контекст устройства становится известен графическим функциям GDI через *дескриптор контекста*, который для всех этих функций служит первым параметром.
- Контекст устройства относится к числу системных ресурсов, количество которых в системе может быть ограничено;
- Работа с такого рода ресурсами всегда протекает одинаково:
 - сначала надо получить у системы требуемый ресурс
 - закончив работу с ним, вернуть его системе.

Вывод изображения в окно

- Таким образом, для того чтобы вывести в окно некоторое изображение, необходимо выполнить следующие действия, последовательность которых, в сущности, определяет алгоритм обработки сообщения WM_PAINT:
 - получить у системы контекст устройства для данного окна;
 - изменить при необходимости режимы рисования или характеристики конкретных инструментов;
 - сформировать с помощью графических функций GDI требуемое изображение;
 - вернуть Windows занятый у нее контекст устройства, приведя его предварительно в исходное состояние.

Все перечисленные действия выполняются в функции OnPaint().

- Для получения контекста устройства предусмотрена функция BeginPaint(), требующая два параметра.
 - Первый параметр представляет собой дескриптор того окна, в котором мы предполагаем рисовать и для которого требуется контекст устройства.
 - Второй параметр - это адрес структурной переменной типа PAINTSTRUCT, которую функция BeginPaint() заполняет некоторыми данными.
- В случае своего успешного выполнения функция BeginPaint() возвращает дескриптор контекста устройства, который имеет тип HDC (Handle of Device Context).

Программа вывода строки

- В нашей программе дескриптор контекста устройства поступает в переменную `hdc`.
- В рассматриваемом простом примере изображение на экране представляет собой просто строку текста; для вывода строки используется функция `GDI TextOut()`,
- `TextOut()`, в качестве первого параметра требует указания дескриптора контекста устройства. Поскольку никакие элементы контекста устройства в программе не изменяются, вывод строки осуществляется шрифтом, действующим по умолчанию.
- Возврат контекста в Windows осуществляется функцией `EndPaint()`, использующей те же аргументы, что и функция `BeginPaint()`.

Структурная переменная типа PAINTSTRUCT

- Функции `BeginPaint()` и `EndPaint()` используют структурную переменную (в нашем случае `ps`) типа `PAINTSTRUCT`. В то же время в программе она никак не используется.
- Что содержит эта структура?

```
void OnPaint(HWND hwnd){  
    char szText[]="Строка текста для вывода в главное окно";  
    PAINTSTRUCT ps;           //Структура, требуемая для рисования в рабочей области  
    HDC hdc=BeginPaint(hwnd,&ps);    //Получение контекста устройства  
    TextOut(hdc,5,30,szText,strlen(szText));    //Вывод строки текста с точки 5,30  
    EndPaint(hwnd,&ps);           //Освобождение контекста устройства  
}           //Конец функции OnPaint()
```

Стр

Элемент структуры `hdc` - это тот же самый дескриптор контекста устройства, который служит для функции `BeginPaint()` возвращаемым значением.

- Структура `PAINTSTRUCT`, заполняемая при обработке сообщения `WM_PAINT`, содержит следующие элементы:

```
typedef struct tagPAINTSTRUCT {  
    HDC hdc;           //Дескриптор выделяемого контекста устройства  
    BOOL fErase;       //Флаг перерисовки фона  
    RECT rcPaint;      //Область вырезки  
    BOOL fRestore;     //Зарезервировано  
    BOOL fIncUpdate;   //Зарезервировано  
    BYTE rgbReserved[32]; //Зарезервировано  
} PAINTSTRUCT;
```

Область вырезки `rcPaint`, которая сама представляет собой структуру типа `RECT`, служащую для описания прямоугольной области.

Флаг перерисовки окна обычно равен нулю.

- Если, однако, в процессе регистрации класса окна не определить элемент `wc.hbrBackground` в структуре `WNDCLASS`, т. е. не задать кисть для закрашивания фона окна, функция

`BeginPaint()` заполнит поле `ps.fErase` нулевым значением.

```
hbrBackground=  
StockBrush(WHITE_BRUSH);
```

В программе это будет означать, что программа должна сама закрашивать окно, которое иначе будет прозрачным.

Практически такой режим используется редко.

Область вырезки rcPaint

```
typedef struct tagRECT {  
    int left;    //x-координата левого верхнего угла прямоугольника  
    int top;     //y-координата левого верхнего угла прямоугольника  
    int right;   //x-координата правого нижнего угла прямоугольника  
    int bottom; //y-координата правого нижнего угла прямоугольника  
} RECT;
```

- Переменные типа RECT чрезвычайно широко используются в программах для Windows, поскольку области экрана, с которыми имеет дело Windows, всегда имеют прямоугольную форму.
- В данном случае переменная ps.rcPaint описывает ту область окна, которая в процессе манипуляций с окном была повреждена и требует перерисовки.
- Рассмотрим этот вопрос более подробно.

