Урок 3. Вывод в окно, контексты отображения, функции GDI

Вывод в окно и его обновление

Рисование на экране, принтере и других устройствах является важнейшим аспектом Windows-приложений. На протяжении всего выполнения Windows-приложение повторно рисует и перерисовывает содержимое своих окон в ответ на действия пользователя и другие события.

Сообщение WM_PAINT

В процессе своей работы Windows-приложение выводит информацию для пользователя в рабочую область окна.

Рабочая, или ***клиентская***, область окна – это внутренняя часть окна, на которой программа может рисовать и представлять визуальную информацию для пользователя. При этом там что-нибудь будет оставаться до тех пор, пока программа опять что-то специально не нарисует. Например, другое окно может перекрыть часть рабочей области окна приложения. Поэтому после того, как часть рабочей области перекрытого окна станет видимой, Windows выдает запрос, требующий, чтобы программа перерисовала эту часть рабочей области.

Замечание. Windows уведомляет приложения о различных событиях путем постановки синхронных сообщений в очередь сообщений приложения (с помощью функции PostMessage) и путем отправки асинхронных сообщений соответствующей процедуре окна (с помощью функции SendMessage). Посылая сообщение WM_PAINT, Windows уведомляет оконную процедуру о том, что часть рабочей области окна необходимо обновить.

Большинство программ Windows вызывают функцию UpdateWindow при инициализации в WinMain, сразу перед входом в цикл обработки сообщений. Windows использует эту возможность для отправки в оконную процедуру первого сообщения WM_PAINT.

Вывод: оконная процедура получает сообщение WM_PAINT, если все окно или его часть помечены как требующие перерисовки. Это происходит в следующих случаях:

- Предварительно скрытая область окна открылась, когда пользователь передвинул окно или выполнил какие-то действия, в результате которых окно вновь стало видимым.
- Пользователь изменил размеры окна (и если в стиле класса окна установлены биты CS_HREDRAW или CS_VREDRAW).
- Программа для прокрутки части рабочей области использовала функцию ScrollWindow или ScrollDC.
- Программа информировала о необходимости перерисовки всего окна или его части при помощи функций InvalidateRect или InvalidateRgn.

Замечание. В последнем случае для немедленной перерисовки этой области программе следует затем воспользоваться функцией UpdateWindow для самостоятельной генерации сообщения WM_PAINT, в противном случае Windows сама сгенерирует сообщение WM_PAINT, но произойдет это только после полной обработки текущего сообщения.

В некоторых случаях, когда часть рабочей области временно закрывается, Windows пытается сначала ее сохранить, а затем восстановить. Обычно это происходит, когда:

- Windows удаляет диалоговое окно или окно сообщения, которое перекрывало часть программы.
- Windows раскрывает пункт горизонтального меню и затем удаляет его с экрана.

Замечание. Когда самостоятельное восстановление рабочей области в этих случаях невозможно, тогда Windows посылает в оконную процедуру сообщение WM_PAINT.

В некоторых случаях Windows всегда сохраняет перекрываемую область, а затем восстанавливает ее. Происходит это всегда, когда:

- Курсор мыши перемещается по рабочей области.
- Пиктограмму перемещают по рабочей области.

Перед тем, как передать сообщение WM_PAINT Windows посылает функции окна сообщение WM_ERASEBKGND. Если функция окна передает обработку этого сообщения функции DefWindowProc, последняя в ответ на это сообщение закрашивает внутреннюю область окна с использованием кисти, указанной в классе окна.

Поэтому, если функция окна нарисует что-либо в окне во время обработки сообщений, отличных от WM_PAINT, то после прихода первого же сообщения WM_PAINT нарисованное изображение будет закрашено.

Работа с сообщением WM_PAINT требует, чтобы программист знал о том, как выводить информацию на экран:

- Windows-программа должна быть структурирована таким образом, чтобы в ней была собрана вся информация, необходимая для рисования в рабочей области, но рисование осуществлялось бы только “по запросу” – когда Windows отправит оконной процедуре сообщение WM_PAINT.
- Если же программе необходимо самой инициировать перерисовку своей рабочей области, она может заставить сгенерировать сообщение WM_PAINT (с помощью функций InvaldateRect и UpdateWindow).

Действительные и недействительные прямоугольники

Хотя оконная процедура должна быть готова в любой момент, при получении сообщения WM_PAINT, перерисовать всю рабочую область, часто бывает необходимо перерисовать только небольшую ее часть (обычно прямоугольную область внутри рабочей зоны окна). Это наиболее очевидно, когда часть рабочей области закрыта диалоговым окном. Перерисовка требуется только для области, вновь открывающейся при удалении окна диалога.

Эту область называют недействительным регионом (invalid region) или регионом обновления (update region). Появление недействительного региона в рабочей области вынуждает Windows поместить сообщение WM_PAINT в очередь сообщений.
Таким образом:

- Windows-приложение получает сообщение WM_PAINT только тогда, когда часть рабочей области окна недействительна, т.е. требует обновления. В Windows для каждого окна поддерживается структура информации о рисовании (paint information structure).
- В структуре информации о рисовании (тип PAINTSTRUCT)содержатся координаты минимально возможного прямоугольника, содержащего недействительную область, а также другая достаточно важная информация.

Если еще один регион рабочей области становится недействительным, Widows рассчитывает новый недействительный регион, который содержит оба региона, и запоминает эту новую информацию в структуре информации о рисовании. Windows не помещает в очередь сообщений сразу несколько сообщений WM_PAINT.

Замечание. Отметим, как оконная процедура может получить координаты недействительного прямоугольника:

- При обработке сообщения WM_PAINT: одновременно с получением дескриптора контекста устройства функцией BeginPaint (через структуру типа PAINTSTRUCT), адрес которой передается в эту функцию).
- Она также может получить эти координаты в любое время, вызвав функцию GetUpdateRect.

Каким образом с недействительных прямоугольников можно снять эту "метку".

- После того, как оконная процедура вызывает функцию BeginPaint при обработке сообщения WM_PAINT, вся рабочая область становится действительной.
- Программа, вызвав функцию ValidateRect, также может сделать действительной любую прямоугольную зону в рабочей области. Если этот вызов делает действительной всю рабочую область, тогда любое сообщение WM_PAINT, стоящее в очереди сообщений, удаляется из нее и не обрабатывается.
- Приложение должно выполнять вывод в окно “централизованно” в функции окна при получении сообщения WM_PAINT.
- При обработке сообщения WM_PAINT для увеличения скорости работы следует использовать координаты области окна, подлежащей обновлению, хотя можно обновить и все окно.
- Используя специальные функции, приложение в любой момент времени может определить любую область окна как подлежащую обновлению и послать самому себе сообщение WM_PAINT.

Получение контекста рабочей области окна

Приложения рисуют на устройствах вывода, используя ***контекст устройства***. Контекст устройства фактически является структурой данных, он связан с конкретным устройством вывода информации, такими как принтер, плоттер или дисплей.

Что касается дисплея, то в данном случае контекст устройства обычно связан с конкретным окном.

Когда программе необходимо начать рисование, она должна получить контекст устройства. После окончания она должна освободить этот контекст (после освобождения дескриптор этого контекста становится недействительным и не должен далее использоваться).

Во время обработки каждого отдельного сообщения программа должна получить и освободить контекста рабочей области окна. Чаще всего, хранить дескриптор этого контекста в промежутке между обработкой различных сообщений не требуется.

В приложениях Windows при подготовке к процессу рисования на экране обычно используют два метода получения дескриптора контекста рабочей области окна.

Первый метод получения дескриптора контекста рабочей области окна

Этот метод используется при обработке сообщения WM_PAINT. Применяются две функции: BeginPaint и EndPaint. Для этих двух функций требуется ***дескриптор окна*** (передаваемый в оконную процедуру как параметр) и адрес переменной типа структуры PAINTSTRUCT, определяемой в оконной процедуре.

Вызов BeginPaint заполняет поля этой структуры, а также возвращает дескриптор контекста рабочей области окна, который должен запоминаться в переменной типа HDC. Вызов функции EndPaint освобождает дескриптор контекста рабочей области окна.

Типовой процесс обработки сообщения WM_PAINT выглядит следующим образом:

```
case WM_PAINT:
{       PAINTSTRUCT ps; HDC hDC;
        hDC=BeginPaint(hWnd,&ps);
        . . . // использование функций GDI для вывода
        EndPaint(hWnd,&ps);
}; return 0;
```

При обработке в оконной процедуре сообщения WM_PAINT вызовы функций BeginPaint и EndPaint должны обязательно вызываться парой.

Если в оконной процедуре сообщения WM_PAINT не обрабатываются, то они должны передаваться в DefWindowProc (при этом ранее недействительный регион просто превращается в действительный). Эта функция обрабатывает сообщение следующим образом:

```
case WM_PAINT:
{       PAINTSTRUCT ps; HDC hDC;
        hDC=BeginPaint(hWnd,&ps); EndPaint(hWnd,&ps);
}; return 0;
```

Нельзя делать в оконной процедуре следующее:

```
case WM_PAINT: return 0;
```

До тех пор, пока в приложении не произойдет вызов функций BeginPaint и EndPaint (или ValidateRect) Windows не сделает эту область действительной. Вместо этого Windows снова отправит в оконную функцию сообщение WM_PAINT (так как Windows всегда помещает сообщение WM_PAINT в очередь сообщений, если часть рабочей области окна недействительна, т.е. требует перерисовки).

Второй метод получения дескриптора контекста рабочей области окна

Этим способом дескриптор контекста рабочей области окна можно получить, если необходимо рисовать в рабочей области при обработке отличных от WM_PAINT сообщений или для получения информации о самом контексте.

Для получения дескриптора контекста устройства в этих случаях необходимо вызвать функцию GetDC, а для его освобождения – функцию ReleaseDC:

```
case СообщениеОтличноеОтWM_PAINT:
{       HDC hDC=GetDC(hWnd);
        . . . // использование функций GDI для вывода
        ReleaseDC(hWnd,hDC);
}; return 0;
```

Также как BeginPaint и EndPaint, функции GetDC и ReleaseDC, следует вызывать парой. Нельзя вызывать GetDC при обработке одного сообщения, а ReleaseDC – при обработке другого.

Замечание. Как правило, вызовы функций GetDC и ReleaseDC используются в ответ на сообщения от клавиатуры или на сообщения от манипулятора мышь. Они позволяют обновлять рабочую область непосредственно в ответ на пользовательский ввод информации.

Контексты устройств

Windows-приложения выводят графику на аппаратные устройства, используя набор независимых от устройств функций. Иначе Windows-приложения нуждались бы в драйверах устройств для различных видеокарт, принтеров и другого графического оборудования.

Независимость от устройств является одним из главных преимуществ графических операционных систем, подобных Windows.

Приложения рисуют на устройствах вывода, вызывая функции интерфейса графических устройств (GDI), передавая им полученный предварительно контекст устройства. Библиотека GDI, содержащая эти функции, в свою очередь вызывает функции специальных для устройств библиотек или драйверов устройств. Драйверы устройств выполняют операции на реальном физическом оборудовании.

Одним из параметров большинства функций рисования является дескриптор контекста устройства. Кроме идентификации устройства контекст устройства также определяет некоторые другие характеристики, включающие:

- Режим преобразования логических координат в реальные физические координаты устройства.
- Используемые объектов рисования (шрифты, перья, кисти) для выполнения необходимых операций.
- Отсечение функций рисования применительно к видимой области.

Контекст устройства полностью определяет характеристики оборудования. Функции системы рисования используют эту информацию для преобразования не зависимых от устройства вызовов рисования в последовательность зависимых от устройства операций, выполняемых с помощью низкоуровневого программного кода драйвера.

Перед использованием контекста устройства его необходимо создать.

- Наиболее общей функцией создания контекста устройства является функция CreateDC. При вызове этой функции указывается устройство, для которого создается контекст, драйвер, физический порт устройства и специфические для устройства параметры инициализации. После использования такого контекста его следует удалить функцией DeleteDC.
- При рисовании на экране (в окне) приложению не требуется создавать контекст устройства, используя CreateDC. Вместо этого оно может получить дескриптор контекста устройства, представляющего рабочую область окна с помощью функции GetDC или целое окно с помощью функции GetWindowDC.

Обычно приложения настраивают атрибуты контекста рабочей области окна в обработчике сообщения WM_PAINT непосредственно перед началом рисования.

При создании контекста устройства по умолчанию устанавливается режим отображения MM_TEXT. В этом режиме используется логическая система координат, полностью эквивалентная физической. По умолчанию начало координат - точка с координатами (0,0) - находится в левом верхнем углу рабочей области окна, ограниченной сверху заголовком, а с других сторон рамкой окна. Ось x направлена слева направо, а ось y - сверху вниз. Каждая единица по оси x или y соответствует одному пикселу экрана.

Контекст устройства также определяет шрифт, который используется при выводе текста в рабочую область. По умолчанию задается так называемый системный шрифт (system font) или SYSTEM_FONT (символическое имя определено во включаемых файлах Windows). Системный шрифт – это шрифт, котрый Windows использует для текста в заголовках окон, меню и окнах диалога. Системный шрифт является пропорциональным, т.е. разные символы имеют разную ширину, и растровым (все символы определяются как пиксельные шаблоны).

Процесс настройки может оказаться длительным, кроме того, может потребоваться восстановление исходного состояние атрибутов контекста.

В программном интерфейсе GDI имеются две функции, которые позволяют сохранить сразу все атрибуты контекста устройства и затем быстро восстановить их. Для сохранения атрибутов контекста следует использовать функцию SaveDC. Значение, возвращаемое этой функцией, необходимо использовать в качестве параметра для функции RestoreDC, восстанавливающей атрибуты контекста устройства.

Общий контекст (контекст рабочей области окна)

Для получения общего контекста приложение должно вызывать функцию BeginPaint (при обработке сообщения WM_PAINT) или GetDC (при обработке других сообщений). При этом при регистрации класса окна в поле стиля класса не должны использоваться значения CS_OWNDC, CS_PARENTDC, CS_CLASSDC.

Контекст, полученный при помощи функции BeginPaint необходимо освободить перед завершением обработки сообщения WM_PAINT, вызвав функцию EndPaint.

Если контекст был получен с помощью функции GetDC, то после завершения процедуры рисования перед выходом из обработчика сообщений следует ***освободить*** полученный контекст, вызвав функцию ReleaseDC.

Каждый раз, когда приложение получает общий контекст, его атрибуты принимают значения ***по умолчанию***. Если перед рисованием атрибуты изменить, в следующий раз при получении общего контекста эти атрибуты вновь примут значения по умолчанию. Поэтому для данного контекста настройку атрибутов нужно выполнять ***каждый*** раз после его получения.

Контекст класса окон

Можно создать такой контекст, который используется всеми окнами, созданными на базе окна. При регистрации такого класса окна нужно указать стиль CS_CLASSDC.

В отличие от общего контекста приложение, однажды получив контекст для класса окна, могут не освобождать его. То есть для контекста этого типа после функции BeginPaint (GetDC) можно не вызывать функцию EndPaint (ReleaseDC). Если же их вызвать, то они ничего делать не будут и сразу вернут управление. Контекст для класса окна можно использовать в тех случаях, когда нежелательно выполнять настройку многочисленных атрибутов после каждого вызова функции BeginPaint (GetDC). Эту настройку можно выполнить только ***один*** раз.

Каждый раз, когда функция окна получает контекст для класса, ей остается выполнить настройку только двух атрибутов – области ограничения и начала системы физических координат устройства вывода. Остальные атрибуты остаются без изменений и не требуют повторной настройки.

Личный контекст

Указав в стиле класса окна значение CS_OWNDC, можно добиться того, что для каждого окна, созданного на базе этого класса, Windows создаст отдельную структуру контекста.

Личный контекст проще в использовании, но оперативная память будет расходоваться менее экономно.

Личный контекст можно, получив один раз, никогда ***не освобождать***. Можно один раз настроить атрибуты контекста после его получения и использовать полученный контекст до завершения работы приложения.

Родительский контекст

Родительский контекст используется для ***дочерних*** окон. Он позволяет дочерним окнам "унаследовать" атрибуты контекста у родительского окна, что во многих случаях упрощает процедуру настойки этих атрибутов.

Для использования родительского контекста в классе, на базе которого создается дочернее окно, перед регистрацией необходимо указать стиль CS_PARENTDC.

Контекст окна

Все описанные выше контексты позволяют рисовать только ***во внутренней области окна***. С помощью функции GetWindowDC приложение может ***получить*** контекст для окна, позволяющий рисовать в ***любом*** месте окна приложения. Приложение обязано освобождать этот контекст сразу же после использования, вызывая функцию ReleaseDC.

Примечание. Особенностью данного контекста является то, что в нем выбрана система координат, начало которой находится в левом верхнем углу окна (не его внутренней области).

Информационный контекст

Если необходимо только получить информацию об устройстве, приложение может создать не обычный, а информационный контекст. Информационный контекст нельзя использовать для рисования, однако он занимает меньше места в памяти.

Информационный контекст создается при помощи функции CreateIC. После использования информационный контекст следует удалить, вызвав функцию DeleteIC.

Контекст для памяти

Контекст памяти является контекстом устройства, представляющего ***растровое изображение*** (bmp). Используя его, приложения могут, например, ***рисовать*** растровые (bitmap) изображения.

Контекст памяти создается совместимым с тем контекстом отображения, в котором будет выполняться вывод на физическое устройство. Для создания совместимого контекста памя-

ти используется функция CreateCompatibleDC. Созданный таким образом контекст удаляется функцией DeleteDC.

Используя несколько контекстов памяти, приложения могут создавать эффект плавной анимации.

Контекст метафайла

Контекст метафайла позволяет записывать независимые от устройства команды GDI в файл и затем проигрывать такой файл на любом физическом устройстве вывода. Файл может находиться в памяти или на диске, в последнем случае его можно использовать для переноса графического изображения в другое приложение.

Для создания контекста метафайла используется функция CreateMetaFile. После выполнения рисования метафайл следует закрыть функцией CloseMetaFile. Этот вызов возвращает дескриптор метафайла, который впоследствии можно использовать в вызовах PlayMetaFile или различных других функций манипулирования метафайлами.

Контекст физического устройства

Вывод изображений на такое устройство, как принтер, может происходить с использованием тех же приемов, что и вывод в окно. Прежде всего, необходимо получить контекст устройства, затем можно вызывать функции GDI для рисования.

В отличие от контекста отображения, контекст физического устройства не получается, а создается при помощи функции CreateDC. Созданный контекст устройства после использования следует удалить (не освободить), вызвав функцию DeleteDC.

Контекст устройства DISPLAY

В некоторых случаях требуется получить контекст отображения, позволяющий рисовать в ***любом*** месте экрана. Такой контекст можно создать при помощи функции CreateDC, указав в качестве имени драйвера строку “DISPLAY”, а в качестве остальных - значение NULL.

Примечания:

- начало системы координат, выбранной в данный контекст, располагается в верхнем левом углу экрана;
- после использования контекст отображения следует удалить, вызвав функцию DeleteDC.
- приложение может получить контекст отображения для всего экрана при помощи функции GetDC(NULL). Контекст, полученный таким способом, следует **освободить** функцией ReleaseDC, передав вместо идентификатора окна значение NULL.

Рисование с помощью функций GDI

Вывод текста

Для вывода текста можно использовать функцию:

```
BOOL TextOut(HDC hdc, int nXStart, int nYStart, LPCSTR lpszString, int cbString);
```

Функция TextOut не имеет параметров, определяющих шрифт, размер букв, цвет фона, цвет букв и т.д. Эти характеристики текста хранятся в **структуре контекста устройства**.

Цвет текста можно задать функцией SetTextColor. Используемая система координат также определяется в контексте устройства.

Для вывода на экран ***нескольких*** строк текста необходимо знать размеры символов. Размеры символа можно получить при помощи функции GetTextMetrics. Для этой функции требуется ***дескриптор контекста устройства***, поскольку ее возвращаемым значением является информация о шрифте, выбранном в данное время в контексте устройства. Функция копирует различные значения ***метрических*** параметров текста в структуру типа TEXTMETRIC.

Структура TEXTMETRIC обеспечивает полную информацию о шрифте, выбранном в данный момент в контексте устройства.

Вертикальный размер шрифта определяется 5-ю величинами. Два поля описывают ширину символа: tmAveCharWidth (усредненная ширина символов строки) и tmMaxCharWidth (ширина самого широкого символа шрифта). Для фиксированного шрифта эти две величины одинаковы.

В качестве примера приведем фрагмент оконной функции, в котором осуществляется вывод строки текста в центре рабочей области окна:

```
char *str = "Выводимая строка";
// получение контекста отображения
HDC hDC = . . .;
// получение размеров рабочей области окна
RECT r;
GetClientRect(hWnd, &r);
// получение информации о текущем шрифте
TEXTMETRIC  tm;
GetTextMetrics(hDC, &tm);
// координаты X и Y
int x = (r.right - tm.tmAveCharWidth * strlen( str ))/2;
int y = (r.bottom - tm.tmHeight)/2;
// вывод строки
TextOut( hDC, x, y, str, strlen( str ));
// освобождение контекста отображения
. . .;
```

Замечание. Кроме рассмотренной функции TextOut программный интерфейс содержит и другие, более сложные функции, предназначенные для вывода текста.

Координаты

Приложения обычно указывают расположение и размер объекта для вывода в системе **логических** координат. Перед тем, как объект физически появится на экране или на принтере, производится ряд вычислений для получения реальных физических координат устройства.

Физические координаты имеют непосредственное отношение к физическому устройству вывода. В качестве единицы измерения длины всегда используется **пиксел**. Если физическое устройство - экран, то физические координаты называются ***экранными*** координатами.

Для определения физического разрешения используется функция GetDeviceCaps. Физическая система координат “привязана” к физическому устройству вывода, поэтому при ее использовании следует учитывать особенности видеоконтроллера.

Недостатки физической системы координат:

1) вертикальное и горизонтальное разрешения зависят от типа видеоконтроллера.

2) физические размера пикселя и отношение высоты и ширины пикселя также зависят от типа видеоконтроллера.

Логические координаты передаются функциям GDI, выполняющим рисование или вывод текста. Используемые единицы измерения зависят от **режима отображения**.

Режим отображения - это атрибут контекста устройства, влияющий на используемую функциями GDI систему координат.

Вывод: при отображении GDI преобразует логические координаты в физические. Способ преобразования зависит от режима отображения и других атрибутов контекста устройства.

Режимы отображения

Приложения Windows могут использовать одну из нескольких логических систем координат, устанавливая соответствующий режим отображения в контексте устройства.

Для установки режима отображения используется функция SetMapMode. Определить режим отображения можно при помощи функции GetMapMode (см. MSDN).

Поддерживаемые GDI режимы отображения с ограничениями:

- MM_TEXT – начало системы логических координат находится в **верхнем левом** углу, и значения вертикальных координат увеличиваются вниз (MM_TEXT эквивалентно отсутствию преобразования вообще). Логическая единица соответствует одному пикселю. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_LOENGLISH – начало отсчета находится в **левом нижнем** углу, и значения вертикальных координат увеличиваются вверх. Логическая единица соответствует 1/100 дюйма. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_HIENGLISH – начало отсчета находится в **левом нижнем** углу, и значения вертикальных координат увеличиваются вверх. Логическая единица соответствует 1/1000 дюйма. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_LOMETRIC – начало отсчета находится в **левом нижнем** углу, и значения вертикальных координат увеличиваются вверх. Логическая единица соответствует 1/10 миллиметра или 1/100 сантиметра. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_HIMETRIC – начало отсчета находится в **левом нижнем** углу, и значения вертикальных координат увеличиваются вверх. Логическая единица соответствует 1/100 миллиметра или 1/1000 сантиметра. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_TWIPS – начало отсчета находится в **левом нижнем** углу, и значения вертикальных координат увеличиваются вверх. Логическая единица соответствует 1/20 пункта или 1/1440 дюйма. Режим не позволяет устанавливать произвольное направление осей координат и произвольный масштаб для осей координат.
- MM_ISOTROPIC – можно выбирать произвольное направление осей координат и произвольный (но одинаковый) масштаб для осей. Приложение может свободно указать начало отсчета логической и физической системы координат, а также их горизонтальные экс-

тенты. Вертикальные экстенты вычисляются GDI, исходя из горизонтальных. Режим не позволяет устанавливать для каждой оси свой собственный масштаб - единственным ограничением является то, что значения вертикальной и горизонтальной логических единиц совпадают.

GDI поддерживает только один режим отображения без ограничений, для которого все сказанное в этом пункте относится полностью – режим MM_ANISOTROPIC.

- MM_ANISOTROPIC - самый универсальный режим. Он позволяет устанавливать произвольное направление осей координат, произвольный масштаб для осей координат, причем для каждой оси можно устанавливать свой собственный масштаб.

Работа с объектами рисования

Преобразование координат определяет расположение объектов на устройстве вывода, в котором они будут выводиться, а их внешний вид определяется с помощью объектов GDI. GDI поддерживает множество **объектов рисования**: перья, кисти, шрифты, палитры, растровые изображения. Приложение при использовании этих объектов должно выполнить следующие действия:

1. Создать объект GDI.
2. Выбрать объект GDI в контекст устройства.
3. Вызвать функцию (или несколько функций) вывода GDI.
4. Восстановить в контексте устройства “старый” объект GDI.
5. Разрушить объект GDI.

Пример программного кода, который использует объект пера (черная линия) для рисования прямоугольника в контексте устройства с дескриптором hDC:

```
// получение контекста отображения
HDC  hDC =. . .;
// создать перо - сплошная зеленая линия
HPEN hPen = CreatePen( PS_SOLID, 1, RGB( 0,255,0 ));
// выбрать перо в контекст отображения
HPEN  hOldPen = ( HPEN )SelectObject( hDC, hPen );
// осуществить вывод изображения
Rectangle( hDC, 0, 0, 100, 100 );
// восстановить в контексте "старое" перо
SelectObject( hDC, hOldPen );
// удалить созданное ранее перо
DeleteObject( hPen );
// освобождение контекста отображения
. . .;
```

Объекты создаются с использованием специальных для каждого типа объектов функций. После создания на объект можно сослаться по дескриптору, который возвращается функцией создания.

Перед использованием объекта в процессе рисования его следует выбрать в контекст устройства с помощью функции SelectObject (палитры выбираются с помощью функции SelectPalette). Функция установки объекта в контексте возвращает дескриптор предыдущего выбора пера, кисти, шрифта или растрового изображения; после завершения рисования этот дескриптор можно использовать для восстановления контекста устройства в начальное состояние.

Неиспользуемые объекты разрушаются с помощью функции DeleteObject.

Приложения могут использовать не только объекты GDI, созданные непосредственно самим приложением, но пользоваться предопределенными системой объектами посредством функции GetStockObject, которую можно использовать для получения дескриптора различных системных перьев, кистей, шрифтов и палитр.

Перья

Перья используются для рисования линий, кривых и контуров других фигур. По умолчанию в контексте рабочей области окна установлено перо в виде черной сплошной линии.

Для получения встроенного пера можно воспользоваться функцией GetStockObject. Однако при помощи встроенных перьев нельзя нарисовать цветные, широкие, штриховые и штрихпунктирные линии.

Для создания собственных перьев нужно воспользоваться функцией CreatePen:

```
HPEN  CreatePen( int  fnPenStyle, int nWidth, COLORREF clrref );
```

При вызове этой функции приложение указывает стиль, ширину и цвет пера. При помощи функции CreatePen нельзя создать перья штрихового и точечного стиля с шириной больше единицы. Однако при помощи функции ExtCreatePen это сделать можно.

Цвет пера определяется как значение RGB; однако при указании соответствующего элемента логической палитры Windows обычно заменяет его ближайшим цветом палитры. Исключение составляет случай, когда ширина пера больше единицы и установлен стиль PS_INSIDEFRAME; в этой ситуации Windows использует цвета с оттенками.

На рисование пером влияет режим смешивания переднего плана. Этот режим устанавливается с помощью функции SetROP2. Различные логические операции между цветом пера и цветом пиксела определяются несколькими установками. Текущий режим смешивания можно получить с помощью функции GetROP2.

Режим фона влияет на заполнение промежутков между штрихами и точками в линиях. Для установки режима фона предназначена функция SetBkMode. Установить цвет фона можно при помощи функции SetBkColor.

По умолчанию в контексте отображения установлен прозрачный режим фона OPAQUE. В этом режиме промежутки закрашиваются цветом фона, определенным как атрибут контекста устройства.

Приложение может установить режим фона TRANSPARENT, в этом случае промежутки в линиях не будут закрашиваться.

Замечание. Небольшое замечание относительно концов широких линий - они у них закруглены. Для изображения широкой линии с прямыми концами следует задать прямоугольную область ограничения. Можно нарисовать широкую линию как закрашенный прямоугольник.

Кисти

Кисти используются для заполнения содержимого рисуемых фигур. Использование кисти определяет цвет и образец заполнения. По умолчанию в контексте рабочей области окна установлена кисть черного цвета и сплошной заливки.

Для выбора одной из встроенных кистей можно воспользоваться функцией GetStockObject, однако в Windows есть только монохромные встроенные кисти.

При создании и использовании кистей используются несколько дополнительных функций:

- Для создания цветной сплошной кисти используется функция CreateSolidBrush.
- Приложение может создать одну из шести кистей штриховки функцией CreateHatchBrush.
- Можно использовать свой собственный стиль штриховки, создав кисть из битового изображения функцией CreatePatternBrush.

При использовании кистей цвет фона и режим фона управляются функциями SetBkColor и SetBkMode соответственно. Соответствующий эффект на заполнение содержимого объектов оказывает также режим смешивания переднего плана, указанный в вызове функции SetROP2.

Специфической проблемой при использовании образцов и штриховки является проблема начала отсчета кисти. Начало отсчета кисти (brush origin) - это атрибут контекста устройства. Он используется для определения координат точки внутри кисти, которая будет служить начальной при закраске внутренней области фигуры или окна. По умолчанию используются координаты (0,0), соответствующие верхнему левому углу кисти (в системе координат, выбранной в контексте устройства по умолчанию).

- Если кисть используется для закраски внутренней области окна, верхний левый угол изображения кисти совмещается с верхним левым углом этой области. Затем изображение этой кисти многократно повторяется (с шагом в 8 пикселов для Windows 95).
- При закраске фигур начальное расположение кисти привязывается не к фигуре, а по-прежнему к верхнему левому углу внутренней области окна. Поэтому при закраске, например, прямоугольника, верхний левый угол кисти может не совпадать с верхним левым углом прямоугольника.
- Приложение может изменить начало отсчета кисти (сдвинуть кисть) при помощи функций SetBrushOrg иUnrealizeObject. Сначала при помощи функции UnrealizeOrg нужно сбросить координаты кисти, а потом функцией SetBrushOrg установить новые значения координат.

Пример создания и использования кисти и пера в контексте устройства. Создадим и выберем в контекст устройства зеленую сплошную кисть и красное перо, затем воспользуемся имя для рисования прямоугольника:

```
HDC hDC =. . .; // получение контекста отображения
HBRUSH GreenBrush = CreateSolidBrush( RGB(0, 255, 0));
HBRUSH OldBrush = SelectObject( hDC, GreenBrush );
HPEN RedPen = CreatePen(PS_SOLID, 1, RGB( 255, 0, 0 ));
HPEN OldPen = SelectObject(hDC,RedPen);
Rectangle( hDC, 20, 20, 100, 50 );
SelectObject( hDC, OldBrush ); DeleteObject( GreenBrush );
SelectObject( hDC, OldPen );   DeleteObject( RedPen );
. . .; // освобождение контекста отображения
```

Шрифты

Перед тем, как приложение сможет вывести какой-либо текст, оно должно выбрать логический шрифт для выводимого текста. Приложения Windows могут:

1) использовать либо один из встроенных шрифтов, получив его функцией GetStockObject;

2) либо создать свой при помощи функции CreateFont, передавая ей требуемые характеристики шрифта.

Приложение заказывает шрифт, описывая его параметры (такие, как размеры символов, семейство шрифтов, наклон относительно горизонтальной оси и т.д.). GDI анализирует запрошенные параметры и подбирает наиболее подходящий шрифт.

Метод создания логических шрифтов, предоставляемый Windows, позволяет реализовать полную независимость от устройств. Вместо того чтобы делать приложения зависимыми от определенных шрифтов, шрифты выбираются на основе их характеристик.

При запросе приложением шрифта через функцию CreateFont, Windows выбирает из множества доступных шрифтов один, наиболее соответствующий запрашиваемым характеристикам. Тем не менее, в функции CreateFont можно указать имя и размер шрифта. При этом Windows попытается выбрать желаемый шрифт, если он доступен в системе.

Замечание. Для получения логического шрифта приложения также могут использовать функцию CreateFontIndirect. Ее параметром является указатель на структуру LOGFONT. Эта функция наиболее эффективна в сочетании со стандартным диалоговым окном “Font” (“Шрифт”), которое возвращает выбор пользователя в форме структуры LOGFONT.

Пример создания и использования шрифта в контексте устройства. Создадим и выберем в контекст устройства логический шрифт: системный шрифт со стандартными установками (системный шрифт – это шрифт, который Windows использует для текста в заголовках окон, меню и окнах диалога):

```
HDC hDC =. . .; // получение контекста отображения
LOGFONT  log_font, old_font; char *str = ”Text String”;
memset( &log_font, 0, sizeof( LOGFONT ));
sprintf( log_font.lfFaceName, "System" );
log_font.lfWeight = FW_NORMAL;
log_font.lfCharSet = ANSI_CHARSET;
log_font.lfOutPrecision = OUT_DEFAULT_PRECIS;
log_font.lfClipPrecision = CLIP_DEFAULT_PRECIS;
log_font.lfQuality = DEFAULT_QUALITY;
log_font.lfPitchAndFamily = DEFAULT_PITCH|FF_MODERN;
log_font.lfOrientation = log_font.lfEscapement=0;
font = CreateFontIndirect( &log_font);
old_font = SelectObject( hDC, font );
SetTextColor( hDC, rgb_text ); SetBkMode( hDC, TRANSPARENT );
TextOut( hDC, 100, 100, str, strlen( str ));
SelectObject( hDC, old_font );
DeleteObject( font );
. . .; // освобождение контекста отображения
```

Битовые (растровые) изображения

Если нужно вывести на экран сложное изображение, содержащее множество мелких деталей, едва ли его имеет смысл рисовать при помощи перьев и кистей.

Можно заранее подготовить растровое изображение в одном из графических редакторов и сохранить его в bmp-файле. Затем можно воспользоваться этим изображением в приложении, включив его в состав ресурсов приложения.

В битовом изображении bitmap каждый пиксел представлен своим цветом. Рисование битовых изображений выполняется путем копирования соответствующего массива графических данных в видеопамять. Для выполнения таких операций приложение обращается к GDI, вызывая соответствующую функцию.

Замечание. Только битовые изображения дают возможность получить на экране красивые рисунки, приближающиеся по качеству к слайдам, а также работать с движущимися изображениями.

Для того чтобы использовать ресурс битового изображения, его нужно предварительно загрузить в память при помощи функции LoadBitmap. После загрузки изображения эта функция выдает дескриптор битового изображения типа HBITMAP. Каждый вызов LoadBitmap должен быть уравновешен вызовом DeleteObject.

После загрузки растрового изображения его можно использовать.

Пример, в котором загруженное уже битовое изображение (его дескриптор - hBitmap) размером 32x32 пикселя; выводится на некоторое устройство, начиная от точки с координатами (100, 100):

```
HBITMAP hBitmap = LoadBitmap( hInst,MAKEINTRESOURCE(ИдентификаторРесурса));
. . .
HDC hDC =. . .; // получение контекста отображения
HDC BitmapDC = CreateCompatibleDC( hDC );
HBITMAP hOldBitmap = SelectObject( BitmapDC, hBitmap );
BitBlt( hDC, 100, 100, 32, 32, BitmapDC, 0, 0, SRCCOPY );
SelectObject ( BitmapDC, hOldBitmap );
DeleteDC( BitmapDC );
. . .
. . .; // освобождение контекста отображения
DeleteObject(hBitmap);
```

Этот код создает еще один контекст устройства, совместимый с контекстом hDC. Это делается при помощи функции CreateCompatibleDC. Прежде чем растровое изображение сможет быть выведено на экран, оно сначала должно быть выбрано в этот второй контекст устройства.

Функция SelectObject выбирает объект в контекст устройства BitmapDC, совместимый с контекстом hDC. Она всегда возвращает объект, который перед этим был в контексте устройства (для последующего восстановления). После того, как растровое изображение показано (функцией BitBlt) и восстановлено предыдущее растровое изображение (функцией SelectObject), нужно удалить второй контекст устройства BitmapDC при помощи DeleteDC.

Примечания:

1. Не забывайте копировать растровое изображение в его собственный, специально созданный контекст устройства.
2. Не забывайте запоминать старое растровое изображение в переменной.
3. Не забывайте копировать это старое растровое изображение обратно в контекст устройства.
4. Не забывайте удалять контекст устройства, специально созданный для растрового изображения.

Вывод: весь полный жизненный цикл растрового изображения включает в себя 7 отдельных шагов. Даже маленькая ошибка в одном из пунктов может привести к неправильным

действиям Windows. В частности, очень важно не забывать вызывать SelectObject или удалять контекст устройства.

Основные этапы работы с растровыми изображениями:

1) загрузить растровое изображение;
2) создать совместимый контекст устройства;
3) передать растровое изображение в совместимый контекст устройства;
4) передать с помощью BitBlt совместимый контекст устройства в контекст устройства окна;
5) выбрать старое растровое изображение в совместимый контекст устройства;
6) удалить совместимый контекст устройства;
7) удалить растровое изображение.

Отсечение

Технология отсечения имеет фундаментальное значение в многозадачной оконной среде. Благодаря этой технологии приложения не могут случайно вывести информацию на дисплей вне клиентской части своих окон, также не существует проблем при перекрытии части окна приложения или попадании его части за пределы экрана.

Кроме использования отсечений системой, приложения получают явный доступ ко многим функциям отсечения. Они могут определить область отсечения для контекста устройства и ограничить вывод графики этой областью.

Область отсечения обычно (но не всегда) является прямоугольной областью. Перечислим различные типы создаваемых областей: овальная область, многоугольная область, прямоугольная область, прямоугольная область со скругленными краями.

Приложение может выбрать созданную область отсечения в контекст устройства, вызвав функцию SelectObject или SelectClipRgn. Существующие области отсечения можно объединять с новыми областями при помощи функции SelectClipRgnExt.

Другая форма отсечения осуществляется при использовании путей отсечения. Путь отсечения может определить сложные фигуры отсечения, которые не могут быть определены через области отсечения.

Операции рисования

Какие операции рисования можно осуществлять с использованием GDI?

Рисование линий и окружностей не сводится к многократному вызову функции рисования точки. Многие функции рисования выполняются драйвером или даже аппаратурой видеоконтроллера, что значительно ускоряет вывод.

Приложения Windows могут рисовать прямые и ломаные линии, а также дуги эллипса или окружности. Параметры этих линий определяются несколькими атрибутами отображения. Это режим отображения, цвет фона, режим фона (прозрачный или нет), режим рисования, цветовая палитра, перо.

Для рисования прямых линий (и только для этого) в контексте отображения хранятся координаты текущей позиции пера. Эту текущую позицию можно изменять, указывая этим начало рисования линии. После работы функции рисования линии текущая позиция перемещается в заданную точку.

Преимущества использования отдельных функций для установки текущей позиции и рисования линии из текущей позиции с последующим изменением текущей позиции проявляются при рисовании ломаных линий.

Помимо линий приложения могут рисовать замкнутые закрашенные или незакрашенные прямоугольники, эллипсы, многоугольники и т.д.

- Для закрашивания внутренней области используется кисть, задаваемая в контексте устройства.
- Внешний контур обводится пером, которое также выбирается в контексте отображения.
- Учитываются и остальные описанные ранее атрибуты.

В интерфейсе GDI есть средства, позволяющие приложения создавать область достаточно сложной формы из прямоугольных, многоугольных и эллиптических областей. Такие области можно закрашивать или использовать в качестве маски при выводе графического изображения. В последнем случае область называется областью ограничения. Созданные области не имеют никакой связи с контекстом устройства. Область - это объект, принадлежащий GDI, поэтому его следует удалить после использования. Лучше всего для этого использовать функцию DeleteObject.

Замечания о печати

GDI также отвечает за обеспечение твердой копии вывода на принтер, плоттер и другие устройства вывода. Вывод на устройства, обеспечивающие твердые копии, ничем не отличаются от вывода на дисплей. Стандартный набор функций GDI вызывается для ***контекста принтера***.

Иногда необходимо узнать физические характеристики выводимой страницы и ограничения устройства, однако, приложения в большинстве случаев могут с минимальными изменениями использовать один и тот же программный код для печати и для вывода на дисплей.

В печать вовлекаются несколько компонентов Windows.

- Основной компонент – это спулер печати, который управляет процессом печати.
- Процессор печати преобразует выгруженную работу печати в вызовы драйвера устройства.
- Драйвер устройства генерирует неструктурированный вывод, который затем обрабатывается самим принтером.
- Монитор порта передает неструктурированные команды устройства на физическое устройство через определенный порт или сетевое соединение.

Замечание. Существует набор специальных функций Win32 для работы печати, получающие информацию о задачах и о принтерах и управляющие процессом печати.

Контрольные вопросы:

1. О чем уведомляется окно, получившее сообщение WM_PAINT?
2. При возникновении каких причин окно получает сообщение WM_PAINT?
3. В каких случаях Windows самостоятельно восстанавливает изображение в рабочей области?
4. Какое сообщение обязательно посылается окну перед отправкой ему сообщения WM_PAINT? Как реагирует на это сообщение оконная функция обработки сообщений по умолчанию?

5. Что произойдет после первого же прихода сообщения WM_PAINT, если функция окна нарисует что-либо в окне во время обработки сообщений, отличных от WM_PAINT?
6. Что такое недействительная область (или областью обновления)?
7. Что происходит при появлении недействительной области окна?
8. Может ли в очереди сообщений находится несколько сообщений WM_PAINT?
9. Как приложение может удалить из очереди сообщение WM_PAINT?
10. При помощи чего приложения могут рисовать на устройствах вывода?
11. Когда приложение может использовать функции BeginPaint и EndPaint?
12. Обязательно или нет передавать необрабатываемые сообщения WM_PAINT в функцию обработки сообщений по умолчанию? Что при этом происходит?
13. Как получить контекст устройства, если необходимо рисовать в рабочей области при обработке отличных от WM_PAINT сообщений?
14. Каким образом приложение может явно потребовать перерисовку всего окна или его части?
15. Что делает функция InvalidateRect?
16. Что является итогом выполнения функции UpdateWindow?
17. Какая функция является “противоположной” для функции InvalidateRect?
18. Что определяет контекст устройства?
19. В каких ситуациях приложение может не создавать контекст устройства вывода?
20. Что такое общий контекст? Как приложение может его получить?
21. Когда для общего контекста следует выполнять настройку атрибутов?
22. Для чего служит контекст класса окна?
23. Следует ли освобождать контекст класса окна после его получения?
24. Когда желательно использовать контекст класса окна?
25. Можно ли для контекста класса окна выполнять настройку большинства атрибутов всего один раз?
26. Какие атрибуты контекста класса окна обязательно следует настраивать после его получения?
27. Когда используется личный контекст? В преимущества и недостатки его использования?
28. Кем используется и что позволяет делать родительский контекст?
29. Какой контекст позволяет осуществлять вывод в нерабочую область окна? Как его получить?
30. Как можно получить информацию об устройстве вывода? Контекст какого типа следует для этого использовать?
31. Какой контекст применятся для представления растрового изображения в памяти? С кем должен быть совместим этот контекст? Как создается такой контекст?
32. Что такое метафайл? Что он позволяет делать? Как получить его контекст?
33. При помощи какого контекста осуществляется вывод изображений на такое устройство, как, например, принтер?
34. При помощи какого контекста приложение может рисовать на всей поверхности экрана?
35. Какие объекты GDI приложение может создавать и использовать в процессе своей работы?
36. Какова последовательность действий, выполняемых приложением, использующим объекты GDI?
37. Что необходимо сделать для того, чтобы функции отрисовки могли использовать тот или иной объект GDI?

38. Что такое предопределенные системой объекты GDI? Как приложение может получить к ним доступ?
39. Для чего применяются перья? Каким образом их можно создавать и использовать?
40. Что такое режим смешивания переднего плана, на что он влияет?
41. Что такое режим фона и цвет фона, что они определяют?
42. Для чего используются кисти? Какие кисти может создать приложение?
43. Что такое начало отсчета кисти? Для чего используется этот атрибут?
44. Как приложение может создать свой шрифт, что оно для этого должно указать функции создания шрифта?
45. Каковы основные моменты работы с растровыми изображениями?
46. Для чего используется технология отсечения?
47. Какую форму может иметь область отсечения?

Задание

Создать приложение picture, обеспечивающее при получении сообщения WM_PAINT вывод некоторого изображения в окно.

Методические указания

- При каждом получении сообщения WM_PAINT изображение должно выводиться в случайном месте окна и иметь случайные значения своих характеристик.
- Псевдокод для отрисовки изображения и структурный тип, описывающий его переменные характеристики, приводятся для каждого варианта задания. Следующие характеристики являются обязательными для всех вариантов: ***местоположение, размер*** и ***цвет*** изображения.
- Список сообщений, обязательных для обработки функцией главного окна: WM_PAINT, WM_DESTROY. Остальные сообщения передать на обработку стандартной оконной функции.

Замечание. Текст выполняемого варианта приложения picture сохранить для дальнейшего его использования в качестве исходного для выполнения последующих заданий.