Обработка сообщений

Ресурсы

- содержат различные элементы:
  иконки, курсоры, шаблоны диалоговых окон, меню и др.
- файл описания ресурсов имеет расширение .rc
- нужна компиляция
- создается файл resource.h

```
#include "resource.h"

wndclass.hIconSm = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON1));
```

Изменение иконки (вариант 1)

- если ресурсов нет, то добавить новый ресурс (Resource Files / Add / Resource… / Icon / Import… и выбрать свою иконку)
- скомпилировать файл ресурсов
- узнать числовую константу (IDI) для иконки (View Code… и найти символьную константу для иконки – например, IDI_ICON1)
- подключить заголовочный файл "resource.h "

```
#include "resource.h"
wndclass.hIconSm = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON1));
```

Изменение иконки (вариант 2)

- если ресурсов нет, то добавить новый ресурс (Resource Files / Add / Resource… / Icon / Import… и выбрать свою иконку)
- скомпилировать файл ресурсов
- узнать числовую константу (IDI) для иконки (зайти в Resource View, в дереве правой кнопкой щелкнуть по нужной иконке View Code… и посмотреть Resource Symbols…)

```
wndclass.hIconSm = LoadIcon(hInst, MAKEINTRESOURCE(101));
```

Сообщения мыши

- начинаются с префикса WM_

WM_LBUTTONDOWN	нажата левая кнопка мыши
WM_RBUTTONDOWN	нажата правая кнопка мыши
WM_LBUTTONUP	отпущена левая кнопка мыши
WM_RBUTTONUP	отпущена правая кнопка мыши
WM_LBUTTONDBLCLK	двойной щелчок левой кнопкой мыши
WM_RBUTTONDBLCLK	двойной щелчок правой кнопкой мыши
WM_MOUSEMOVE	перемещение курсора мыши
WM_MOUSEWHEEL	прокрутка колесика

Особенности сообщений мыши

- обрабатываются в функции WndProc()
- в параметре lParam содержатся координаты курсора мыши
- чтобы извлечь координаты курсора мыши, нужны макросы LOWORD (x) и HIWORD (y)
- в параметре wParam содержится информация, какая кнопка мыши нажата
- для обработки двойных щелчков должен быть выставлен стиль класса окна CS_DBLCLKS

Обработка сообщений мыши (пример)

```
int x, y;

switch(iMsg) {
	case WM_LBUTTONDOWN:
		x = LOWORD(lParam);
		y = HIWORD(lParam);
		MoveWindow(hwnd, x, y, 300, 300, true);
		break;
	case WM_DESTROY:
		PostQuitMessage(0);
		return 0;
}
```

Сообщения клавиатуры

- начинаются с префикса WM_

WM_KEYDOWN	нажата любая клавиша без Alt
WM_KEYUP	отпущена нажатая несистемная клавиша
WM_SYSKEYDOWN	нажата любая клавиша в сочетании с Alt
WM_SYSKEYUP	отпущена системная клавиша

- обрабатываются в функции WndProc()
- в параметре wParam содержится виртуальный код нажатой / отпущенной клавиши

Виртуальные коды клавиш

VK_SHIFT	Shift
VK_CONTROL	Control
VK_MENU	Alt
VK_ESCAPE	Esc
VK_F1 … VK_F12	F1 … F12
VK_CAPITAL	Caps Lock
VK_RETURN	Enter
VK_SPACE	Пробел
VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN	Стрелки
0x30 … 0x39	0 … 9
0x41 … 0x5a	A … Z

Сообщения клавиатуры (нажатие символьных клавиш)

- нажатая символьная клавиша формирует сообщение WM_CHAR
- в параметре wParam хранится код символа ASCII (A – виртуальная клавиша A; z – виртуальная клавиша z и т. п.)
- параметр lParam совпадает с lParam аппаратного сообщения: LOWORD(lParam) – количество повторов при удержании клавиши, HIWORD(lParam) – много дополнительной информации

Обработка сообщений клавиатуры (пример)

```
switch(iMsg) {
    case WM_KEYDOWN:
        switch(wParam)  {
            case VK_F1:
                MessageBox(hwnd, L"F1", L"!!!", MB_OK);
                break;
            default:
                break;
        }
        break;
    case WM_CHAR:
        if(wParam == 'a')  {
            MessageBox(hwnd, L"a", L"!!!", MB_OK);
         }
    break;
}
```

Событие WM_PAINT

запрос на перерисовку окна, чтобы сделать недействительную область действительной

Когда наступает:

- окно изменило размер
- использована полоса прокрутки
- стала видимой скрытая область
- вызвана функция InvalidateRect()

Обработка события WM_PAINT

- создать объект структуры типа RECT
- создать объект структуры типа PAINTSTRUCT
- получить дескриптор контекста устройства HDC через вызов функции BeginPaint
- нарисовать в окне все, что нужно
- освободить контекст устройства с помощью функции EndPaint

Структура RECTANGLE

- структура для хранения четырех координат
- координаты окна определяются с помощью функции GetWindowRect(hwnd, &rect)
- координаты рабочей области определяются с помощью функции GetClientRect(hwnd, &rect)

```
struct RECT{
    LONG    left;           // левая координата
    LONG    top;            // верхняя координата
    LONG    right;          // правая координата
    LONG    bottom;         // нижняя координата
};
```

Структура PAINTSTRUCT

- содержит поля, нужные для рисования
- заполняется Windows, когда вызывается функция BeginPaint

```
struct PAINTSTRUCT {
    HDC         hdc;        // дескриптор контекста устройства
    BOOL        fErase;     // обновить фон недействит. области
    RECT        rcPaint;    // координаты недействит. области
    BOOL        fRestore;
    BOOL        fIncUpdate;
    BYTE        rgbReserved[32];
};
```

Подготовка к рисованию (пример)

```
HDC hdc;                    // дескриптор контекста устройства
PAINTSTRUCT ps;             // структура для рисования
RECT rect;                  // структура для координат

switch(iMsg) {
   case WM_PAINT:
      hdc = BeginPaint(hwnd, &ps);      // получение hdc
      GetClientRect(hwnd, &rect);       // получение координат
      // Рисование
      EndPaint(hwnd, &ps);              // освобождение hdc
      break;
   }
/* … */
}
```

Рисование текста

```
int DrawText(HDC hdc, LPCWSTR lpchText, int cchText,
                                        LPRECT lprc, UINT format);
```

```
BOOL TextOut(HDC hdc, int nxStart, int nyStart, LPCTSTR
                                        lpString, int  cchString);
```

- текст рисуется в окне
- нужен контекст устройства (структура данных, которая связана с устройством вывода – принтером, окном и т.п.)

DrawText

```
int DrawText(HDC hdc,     // контекст устройства
             LPCWSTR lpchText,  // рисуемый текст
             int cchText,      // длина текста
             LPRECT lprc,    // размеры поля форматирования
             UINT format);  // формат вывода
```

- если параметр cchText равен -1, то строка должна быть занулена
- некоторые флаги: **DT_CENTER**, **DT_LEFT**, **DT_VCENTER**, **DT_SINGLELINE**, **DT_WORDBREAK**

TextOut

```
BOOL TextOut(HDC hdc,    // контекст устройства
             int nxStart,  // координата x начала рисования
             int nyStart,  // координата y начала рисования
             lpString,     // выводимая строка
             int  cchString);  // размер строки
```

- Выравнивание текста можно задать отдельно с помощью функции SetTextAlign(HDC hdc, UINT textMode)

Рисование текста (пример 1)

```
HDC hdc;
PAINTSTRUCT ps;
TEXTMETRIC tm; // Структура с настройками шрифта
static int height;
switch(iMsg){
   case WM_CREATE:
      hdc = GetDC(hwnd);
      GetTextMetrics(hdc, &tm);
      height = tm.tmExternalLeading + tm.tmHeight;
      ReleaseDC(hwnd, hdc);
      break;
   case WM_PAINT:
      hdc = BeginPaint(hwnd, &ps);
      TextOut(hdc, 10, 10, L"Это текстовая строка", 20);
      TextOut(hdc, 10, height + 10, L"Это текстовая строка", 20);
      EndPaint(hwnd, &ps);
      break;  /* … */
   }
```

Рисование текста (пример 2)

```
HDC hdc;
PAINTSTRUCT ps;
RECT rect;
/* … */
case WM_PAINT:
      hdc = BeginPaint(hwnd, &ps);
      GetClientRect(hwnd, &rect);
      SetTextColor(hdc, RGB(50, 50, 150)); // Цвет шрифта
      SetBkMode(hdc, TRANSPARENT);      // Не закрашивать фон
                                        // под текстом
      DrawText(hdc, L"Hello, Step\nBye, Step", -1, &rect,
                  DT_CENTER | DT_TOP );
      EndPaint(hwnd, &ps);
      break;
   }
```

Рисование вне WM_PAINT (при сообщениях мыши или клавиатуры)

- создать объект структуры типа RECT
- создать объект структуры типа PAINTSTRUCT
- получить дескриптор контекста устройства HDC через вызов функции GetDC()
- нарисовать в окне все, что нужно
- обновить окно
- освободить контекст устройства с помощью функции ReleaseDC() при обработке того же события

Полезные функции (1)

BOOL SetWindowText(HWND hwnd, LPCTSTR str);

Изменяет текст заголовка окна на переданный

```
case WM_LBUTTONDOWN:
    SetWindowText(hWnd, _TEXT("Левая кнопка"));
    break;
case WM_RBUTTONDOWN:
    SetWindowText(hWnd, _TEXT("Правая кнопка"));
    break;
```

Полезные функции (2)

```
BOOL MoveWindow(HWND hwnd, int x, int y,
int iWidht, int iHeight, BOOL bRepaint);
```

Позволяет изменить размер окна или его положение

```
case WM_LBUTTONDOWN:
    MoveWindow(hWnd, 0, 0, rect.right, rect.bottom, true);
    break;
case WM_RBUTTONDOWN:
    MoveWindow(hWnd, 0, 0, rect.right / 2, rect.bottom / 2,
                                                    true);
    break;
```

Вопросы?