Кафедра інформаційних та комп'ютерних систем

Системне програмування

2018

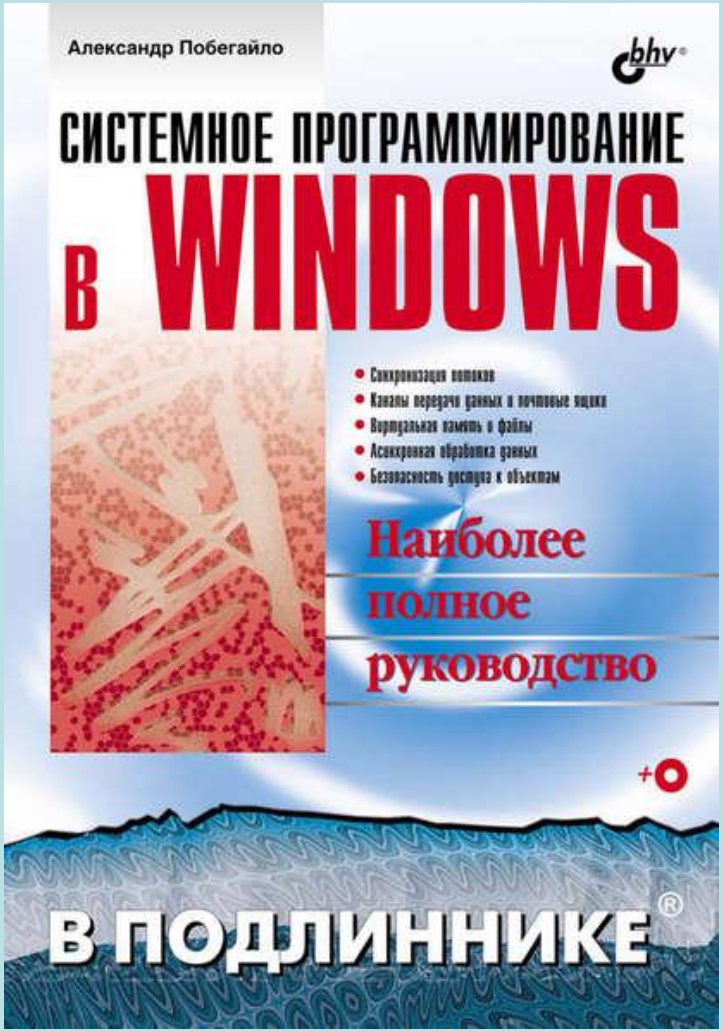
Рекомендуемая литература

"Системное программирование в среде Windows"

(Д. Харт),

В книге описывается разработка приложений с использованием интерфейса прикладного программирования (Application Programming Interface, API) операционных систем Windows компании Microsoft, причем основное внимание уделяется базовым системным службам, включая управление файловой системой, процессами и потоками, межпроцессное взаимодействие, сетевое программирование и синхронизацию

Рекомендуемая литература

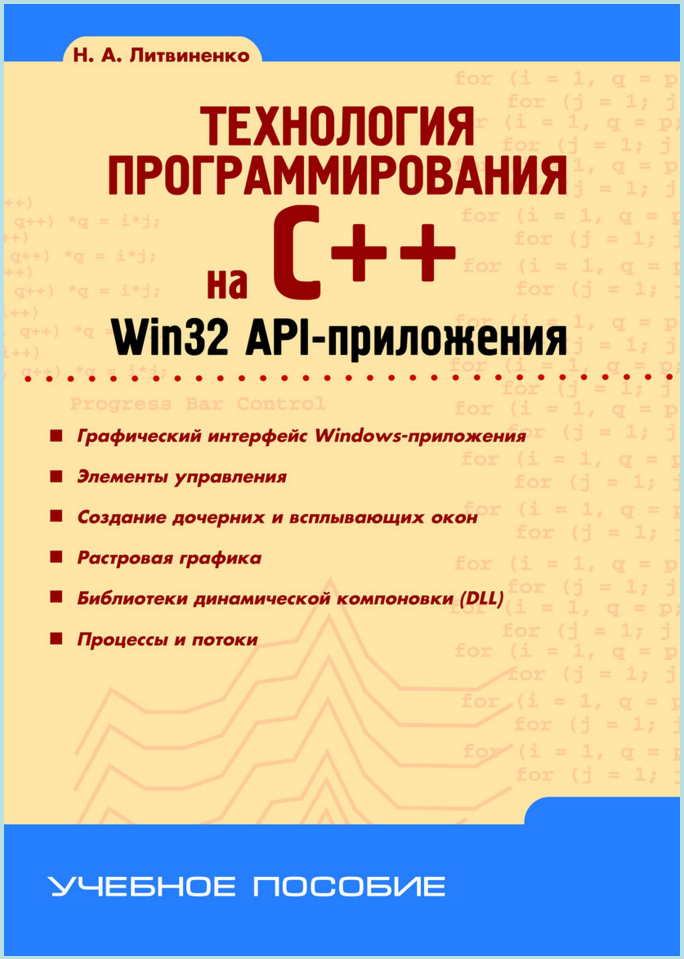


"Системное программирование в Windows"
(А. Побегайло)

Подробно рассматриваются вопросы системного программирования с использованием интерфейса Win 32 API . Описываются управление потоками и процессами, включая их диспетчеризацию; синхронизация потоков; передача данных между процессами, с использованием анонимных и именованных каналов, а также почтовых ящиков; структурная обработка исключений; управление виртуальной памятью; управление файлами и каталогами;

асинхронная обработка данных; создание динамически подключаемых библиотек; разработка сервисов.

Рекомендуемая литература



Литвиненко Н. А.

Технология программирования на C++. Win32 API-приложения. — СПб.: БХВ-Петербург, 2010. — 288 с.: ил. — (Учебное пособие)

Изложен начальный курс низкоуровневого программирования на C++ для Windows с использованием библиотеки Win32 API.

Рассмотрены графический интерфейс Windows-приложения, стандартные диалоговые окна, элементы управления, растровая графика,DLL-библиотеки, процессы и потоки.

Системное программирование

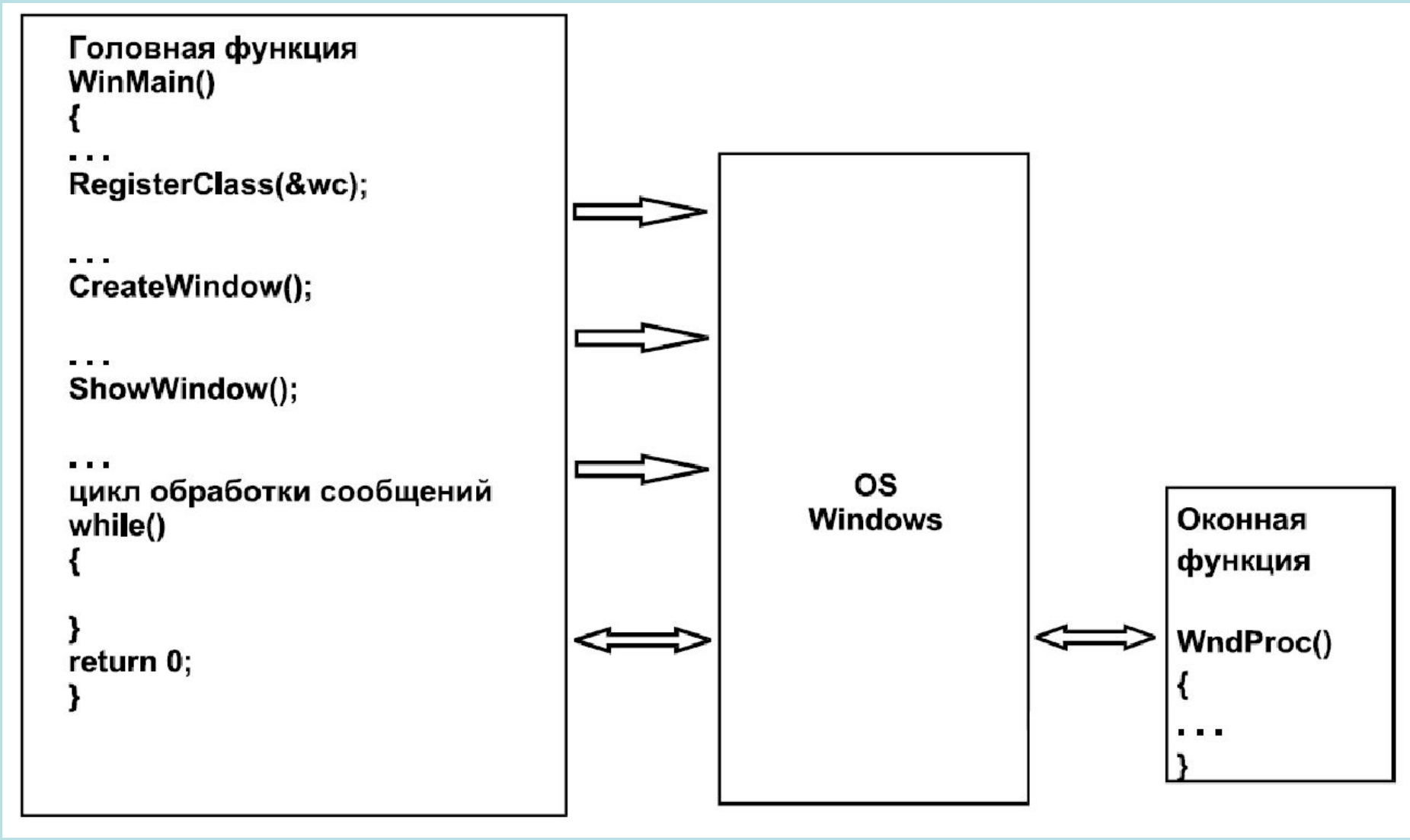
Когда говорят о системном программировании, в большинстве случаев подразумевают разработку программ, имеющих один из трех признаков:

1. пользователем разработанного программного обеспечения является программист. Иными словами, системный программист разрабатывает программное обеспечение для других программистов;
2. разрабатываемое программное обеспечение является повторно используемым и, как правило, оформляется в виде библиотек функций и применяется в нескольких прикладных приложениях. Можно сказать, что системный программист разрабатывает общее, универсальное программное обеспечение;
3. системное программное обеспечение напрямую использует системные вызовы операционной системы.

Интерфейс Windows-приложения

Стиль программирования Windows-приложений принципиально отличается от того, который сложился в операционных системах раннего поколения. В MS-DOS программа монопольно владеет всеми ресурсами системы и является инициатором взаимодействия с операционной системой. Совсем иначе дело обстоит в операционной системе Windows, которая строилась как многозадачная, и именно операционная система является инициатором обращения к программе. Все ресурсы Windows являются разделяемыми, и программа, в дальнейшем будем называть ее *приложением*, не может владеть ими монопольно. В связи с такой идеологией построения операционной системы приложение должно ждать посылки сообщения операционной системы и лишь после его получения выполнить определенные действия, затем вновь перейти в режим ожидания очередного сообщения.

Диаграмма типичного Windows-приложения



Каркас Windows-приложения

В отличие от программы, выполняемой в операционной системе MS-DOS, даже для создания простейшего приложения под Windows придется проделать намного больше работы. Чтобы иметь возможность работать с оконным интерфейсом, заготовка или каркас Windows-приложения должна выполнить некоторые стандартные действия:

1. Определить *класс окна*.
2. Зарегистрировать окно.
3. Создать окно данного класса.
4. Отобразить окно.
5. Запустить цикл обработки сообщений.

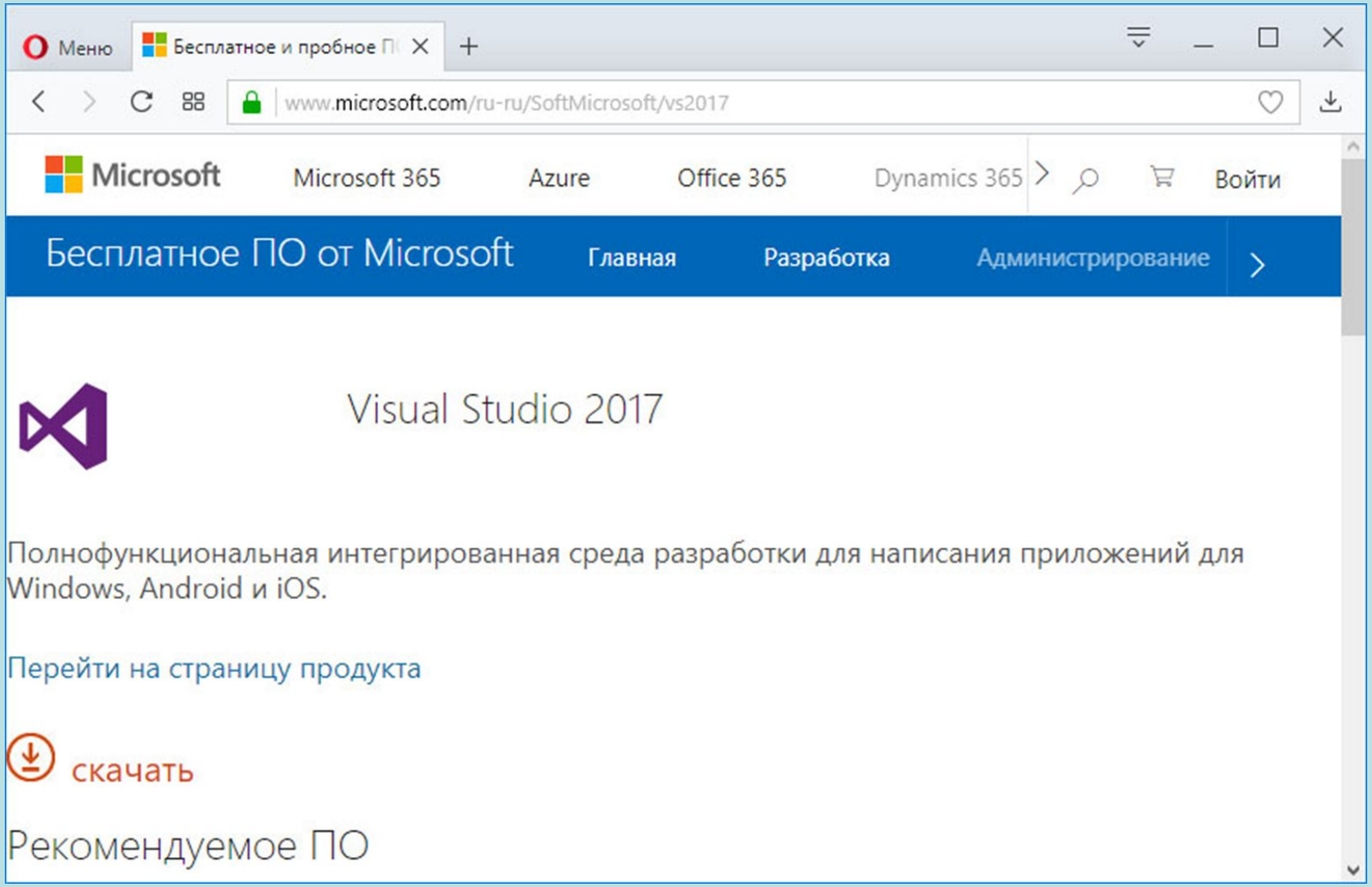
Разработчиками операционной системы Windows была создана библиотека функций, при помощи которых и происходит взаимодействие приложения с операционной системой, так называемые функции *Программного интерфейса приложений*

(*Application Program Interface, API*)

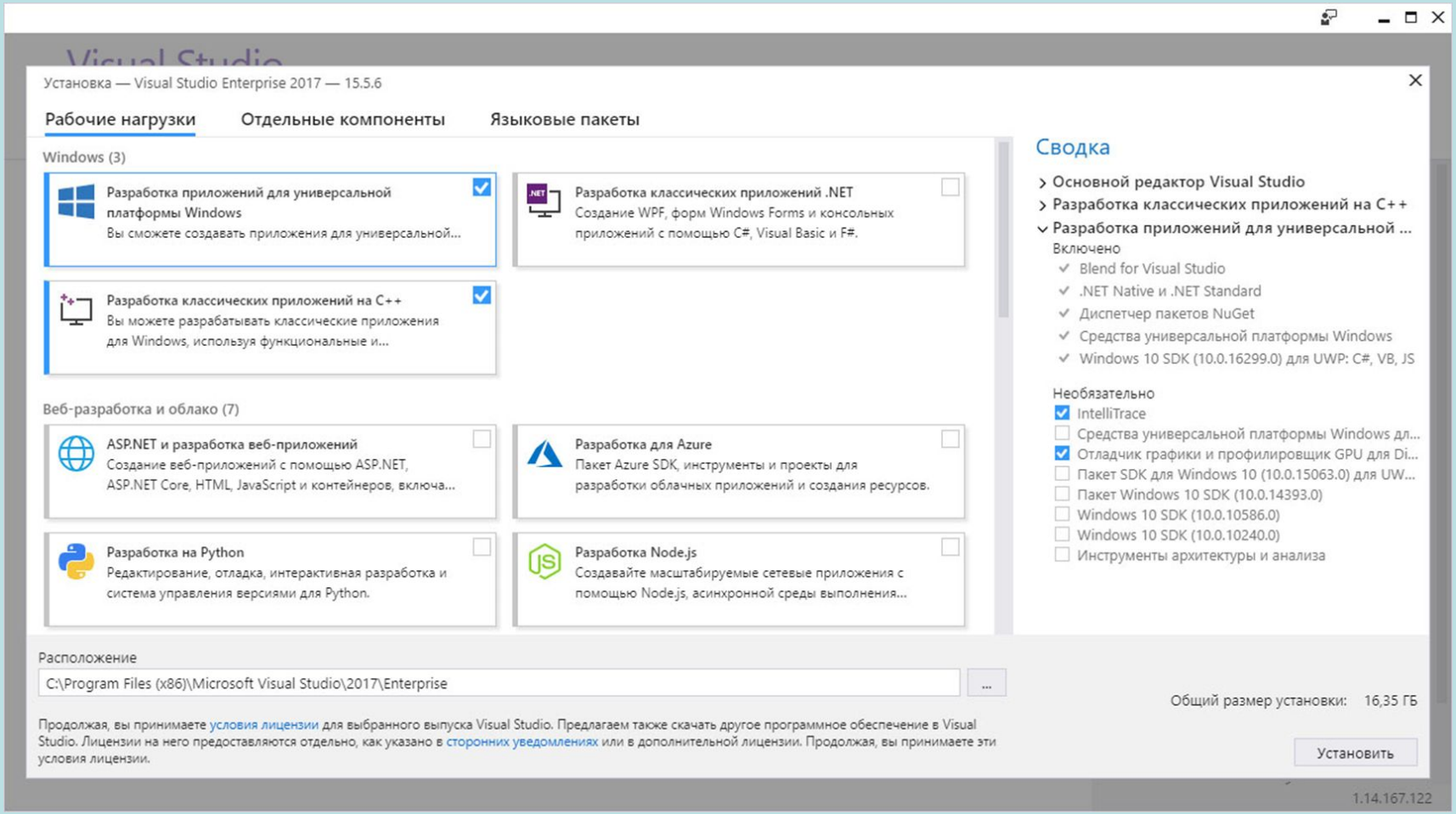
Интерфейс прикладного программирования

Интерфейс прикладного программирования (Application Programming Interface) — набор готовых констант, структур и функций, используемых при разработке программных приложений и обеспечивающих правильное взаимодействие между приложением и операционной системой. В операционной системе Windows интерфейс прикладного программирования называют WinAPI.

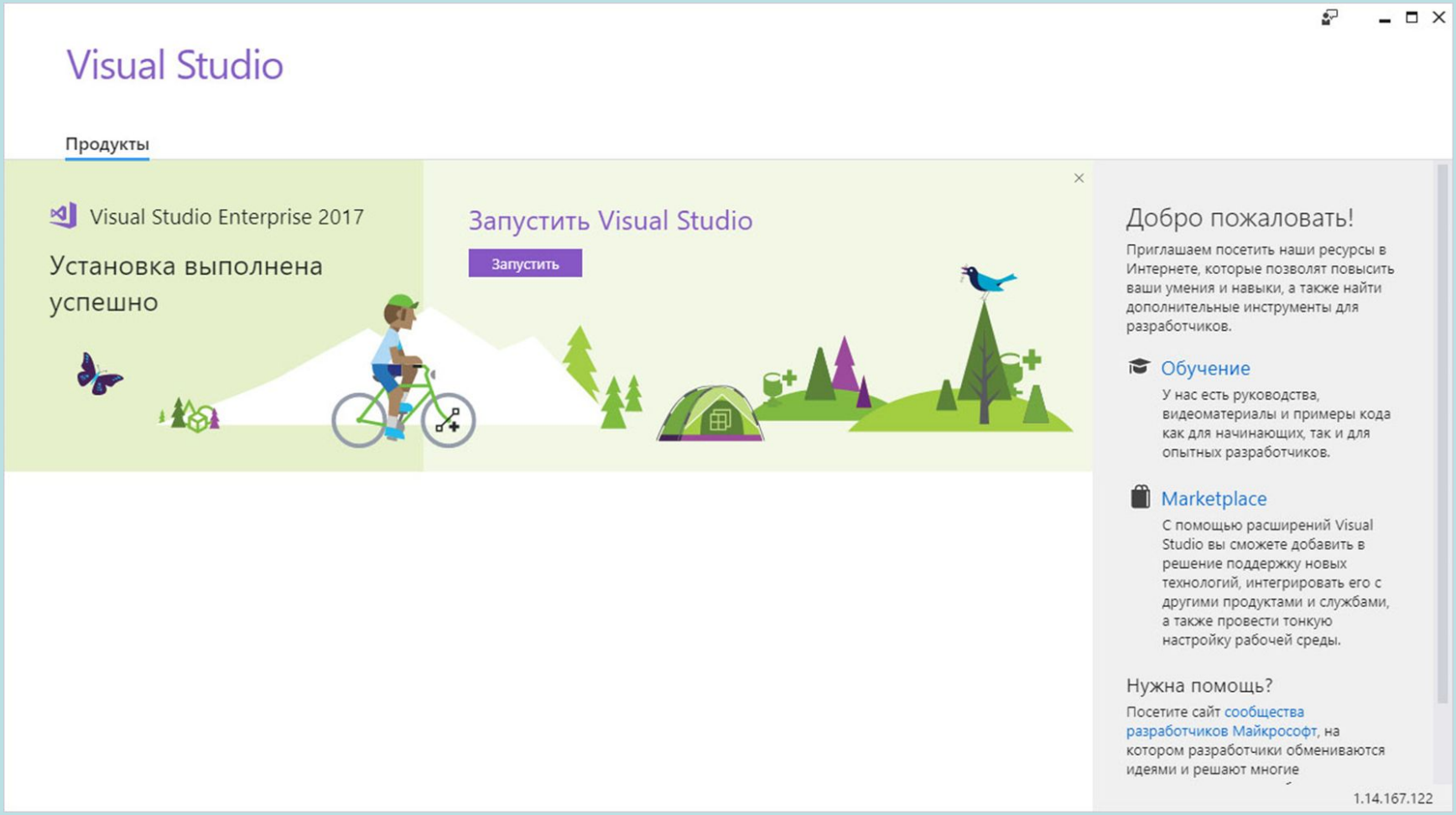
Создание проекта в среде Visual Studio



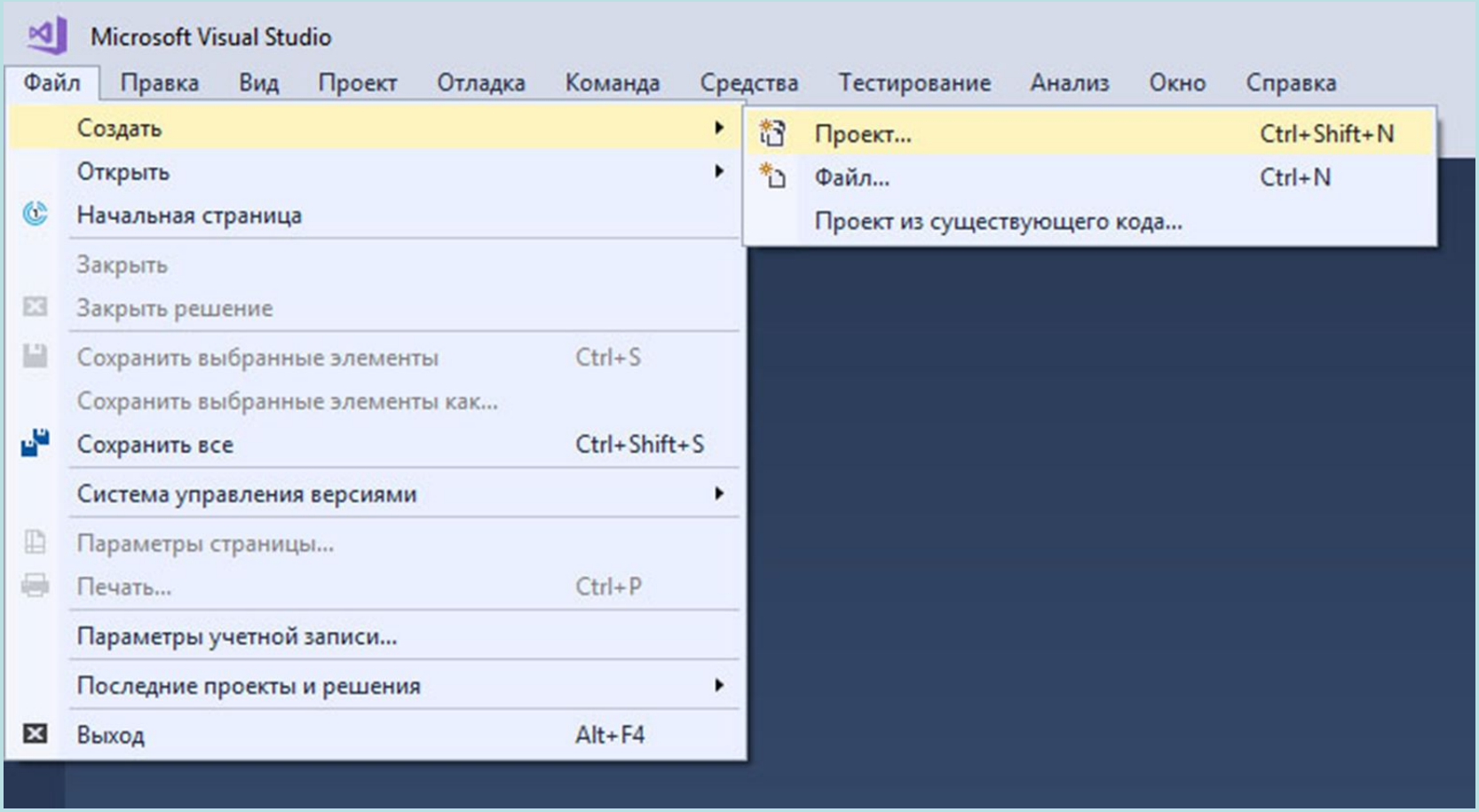
Создание проекта в среде Visual Studio

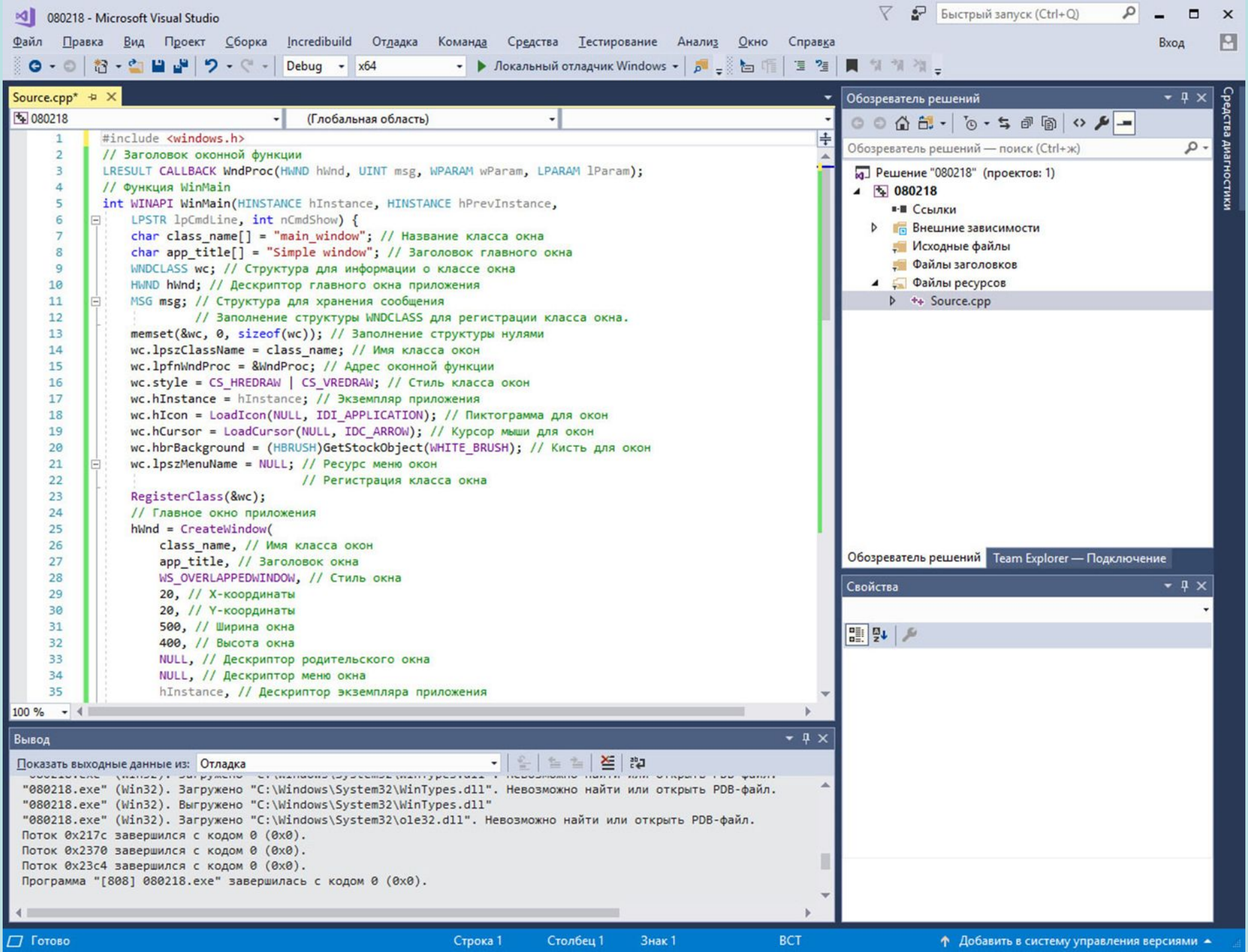


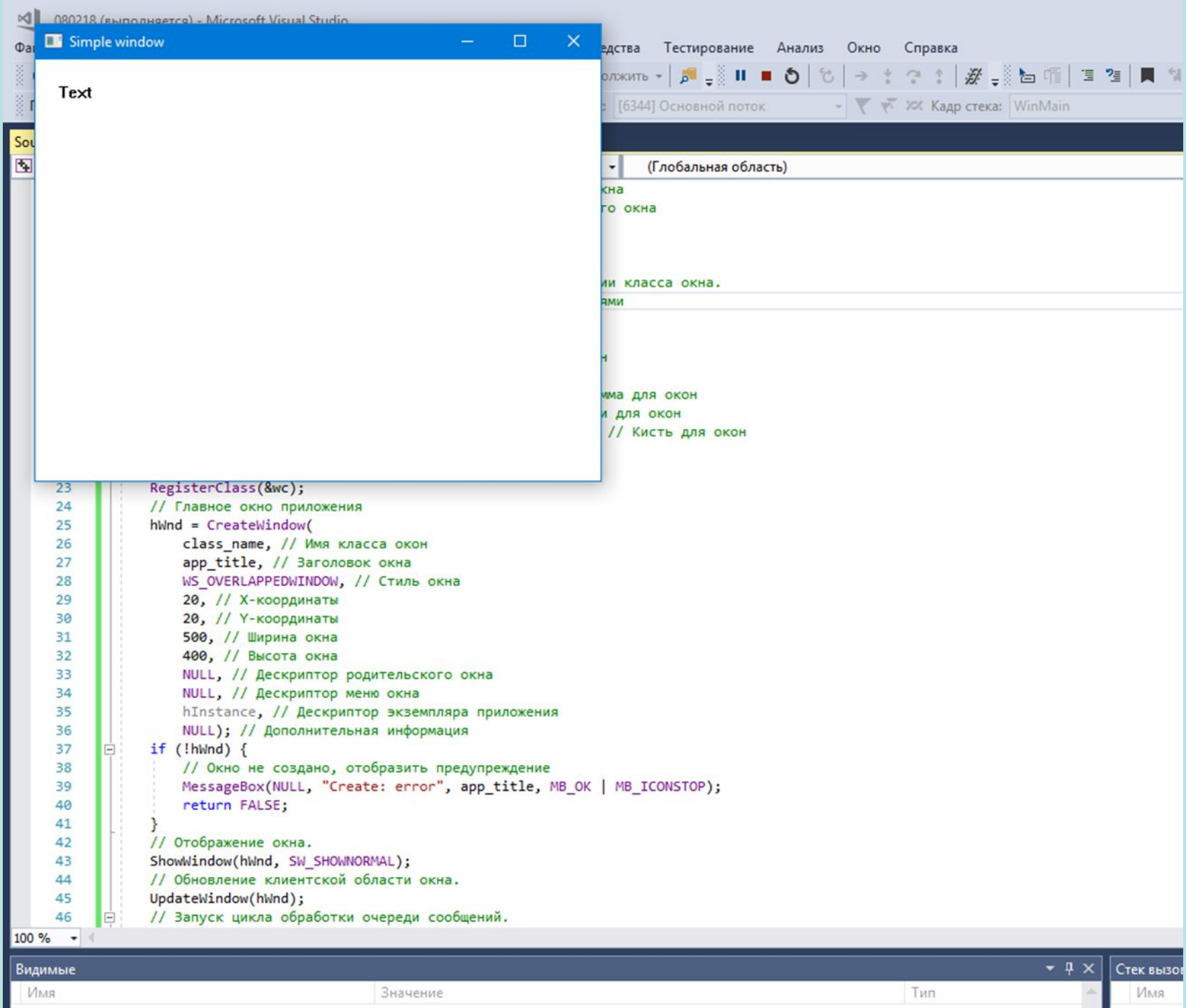
Создание проекта в среде Visual Studio



Создание проекта в среде Visual Studio

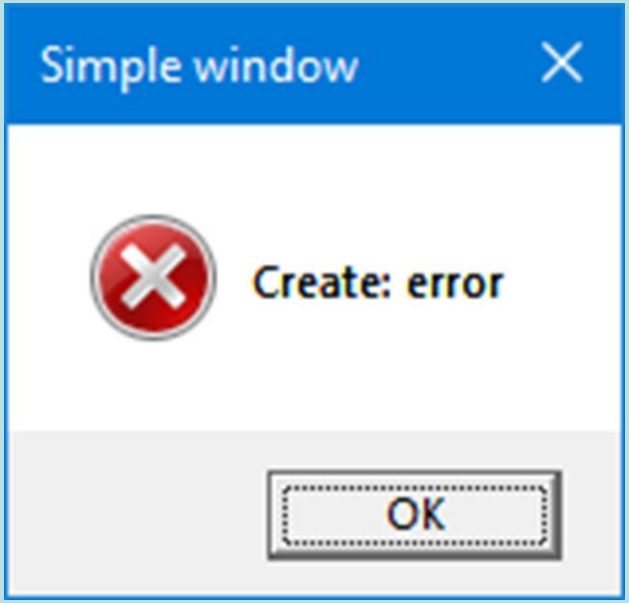



080218 - Microsoft Visual Studio
Быстрый запуск (Ctrl+Q)
Файл Правка Вид Проект Сборка Incredibuild Отладка Команда Средства Тестирование Анализ Окно Справка
Вход
Debug
x64
Локальный отладчик Windows
Source.cpp*
080218
(Глобальная область)
#include <windows.h>
// Заголовок оконной функции
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);
// Функция WinMain
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow) {
char class_name[] = "main_window"; // Название класса окна
char app_title[] = "Simple window"; // Заголовок главного окна
WNDCLASS wc; // Структура для информации о классе окна
HWND hWnd; // Дескриптор главного окна приложения
MSG msg; // Структура для хранения сообщения
// Заполнение структуры WNDCLASS для регистрации класса окна.
memset(&wc, 0, sizeof(wc)); // Заполнение структуры нулями
wc.lpszClassName = class_name; // Имя класса окон
wc.lpfnWndProc = &WndProc; // Адрес оконной функции
wc.style = CS_HREDRAW | CS_VREDRAW; // Стиль класса окон
wc.hInstance = hInstance; // Экземпляр приложения
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // Пиктограмма для окон
wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Курсор мыши для окон
wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); // Кисть для окон
wc.lpszMenuName = NULL; // Ресурс меню окон
// Регистрация класса окна
RegisterClass(&wc);
// Главное окно приложения
hWnd = CreateWindow(
class_name, // Имя класса окон
app_title, // Заголовок окна
WS_OVERLAPPEDWINDOW, // Стиль окна
20, // X-координаты
20, // Y-координаты
500, // Ширина окна
400, // Высота окна
NULL, // Дескриптор родительского окна
NULL, // Дескриптор меню окна
hInstance, // Дескриптор экземпляра приложения
100 %
Обозреватель решений
Обозреватель решений — поиск (Ctrl+ж)
Решение "080218" (проектов: 1)
080218
Ссылки
Внешние зависимости
Исходные файлы
Файлы заголовков
Файлы ресурсов
Source.cpp
Обозреватель решений
Team Explorer — Подключение
Свойства
Средства диагностики
Вывод
Показать выходные данные из: Отладка
"080218.exe" (Win32). Загружено "C:\Windows\System32\WinTypes.dll". Невозможно найти или открыть PDB-файл.
"080218.exe" (Win32). Выгружено "C:\Windows\System32\WinTypes.dll"
"080218.exe" (Win32). Загружено "C:\Windows\System32\ole32.dll". Невозможно найти или открыть PDB-файл.
Поток 0x217c завершился с кодом 0 (0x0).
Поток 0x2370 завершился с кодом 0 (0x0).
Поток 0x23c4 завершился с кодом 0 (0x0).
Программа "[808] 080218.exe" завершилась с кодом 0 (0x0).
Готово
Строка 1
Столбец 1
Знак 1
ВСТ
Добавить в систему управления версиями


Simple window
Text
Тестирование
Анализ
Окно
Справка
[6344] Основной поток
Кадр стека:
WinMain
(Глобальная область)
RegisterClass(&wc);
// Главное окно приложения
hWnd = CreateWindow(
class_name, // Имя класса окон
app_title, // Заголовок окна
WS_OVERLAPPEDWINDOW, // Стиль окна
20, // X-координаты
20, // Y-координаты
500, // Ширина окна
400, // Высота окна
NULL, // Дескриптор родительского окна
NULL, // Дескриптор меню окна
hInstance, // Дескриптор экземпляра приложения
NULL); // Дополнительная информация
if (!hWnd) {
// Окно не создано, отобразить предупреждение
MessageBox(NULL, "Create: error", app_title, MB_OK | MB_ICONSTOP);
return FALSE;
}
// Отображение окна.
ShowWindow(hWnd, SW_SHOWNORMAL);
// Обновление клиентской области окна.
UpdateWindow(hWnd);
// Запуск цикла обработки очереди сообщений.
100 %
Видимые
Имя
Значение
Тип
Стек вызов

```
61          // Необходимо обновить содержимое клиентской области окна
62      case WM_PAINT:
63          HDC hDC;
64          PAINTSTRUCT ps;
65          hDC = BeginPaint(hWnd, &ps); // Получить контекст окна
66          TextOut(hDC, 20, 20, str, strlen(str)); // Нарисовать текст
67          EndPaint(hWnd, &ps); // Освободить контекст окна
68          break;
69          // Нажата левая клавиша мыши в клиентской области окна
70      case WM_LBUTTONDOWN:
71          MessageBox(NULL, "Mouse click", "Message", MB_OK | MB_ICONINFORMATION);
72          break;
73          // Закрытие окна.
74      case WM_DESTROY:
75          // В очередь сообщений приложения помещается сообщение WM_QUIT
76          PostQuitMessage(0);
77          break;
78          // Необработанные сообщения передаются в стандартную
79          // функцию обработки сообщений по умолчанию
80      default: return DefWindowProc(hWnd, msg, wParam, lParam);
81      }
82      return 0;
83  }
```

```
37      if (!hWnd) {
38          // Окно не создано, отобразить предупреждение
39          MessageBox(NULL, "Create: error", app_title, MB_OK | MB_ICONSTOP);
40          return FALSE;
41      }
```



Окна и сообщения

В основе системы Windows лежит механизм обработки сообщений, который транслирует практически каждое событие (нажатие клавиши, перемещение мыши...) в сообщение.

Типичное приложение построено на основе цикла сообщений, который принимает сообщения и перенаправляет их к соответствующим функциям – обработчикам.

Хотя сообщения передаются приложениям, они адресованы не им, а другим компонентам операционной системы – окнам.

Окно

Окно – представляет некую абстрактную сущность, через которую взаимодействуют пользователь и компьютер. Все окна находятся в иерархической зависимости: некоторые из них являются окнами верхнего уровня, другие подчинены своим родительским окнам. В Windows существует множество типов окон. Самыми очевидными типами окон являются большие прямоугольные окна, связанные с приложениями, которые можно переместить, изменить их размеры, минимизировать и т. д. Многие элементы, отображаемые внутри главного или диалогового окна, сами по себе являются окнами. Каждая кнопка, поле ввода, полоса прокрутки, окно списка, пиктограмма, даже фон экрана обрабатывается операционной системой как окно.

Классы окон

Основное поведение окна определяется классом окна. Класс окна несет информацию о начальном внешнем виде окна, пиктограмме по умолчанию, курсоре и ресурсе меню, связанном с окном, а также – адрес оконной функции.

Когда приложение обрабатывает сообщения, оно обычно делает это посредством вызова функции DispatchMessage() для каждого принятого сообщения - это происходит в цикле обработки сообщений. В свою очередь, функция DispatchMessage() вызывает соответствующую оконную функцию, проверяя, для какого класса окна предназначено сообщение. Именно эта оконная функция обрабатывает сообщение, переданное окну.

Классы окон

Существует множество стандартных классов окон, предусмотренных самой операционной системой Windows. Эти системные глобальные классы реализуют функциональные возможности общих элементов управления. Любое приложение может использовать эти классы в своих окнах, например можно реализовать элемент управления – поле ввода, используя класс окна Edit. Приложение также может определять собственные классы окон с помощью функции RegisterClass(). Эта функция позволяет программисту реализовать поведение окна, не являющегося частью ни одного из поддерживаемых системой глобальных классов. Например, таким образом, приложение реализует функциональные возможности своего главного окна, регистрирует пиктограмму окна и ресурса меню.

Точка входа в программу

Точкой входа программы для Windows является функция WinMain(), которая всегда определяется следующим образом:

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

Эта функция использует последовательность вызовов WINAPI и при завершении, возвращает операционной системе Windows целое число.

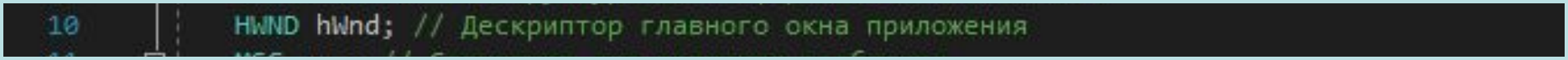
Функция WinMain() выполняет следующие действия:

• сохранение дескриптора экземпляра приложения в переменной;
• регистрация класса окна приложения и другие инициализации;
• создание главного окна приложения и, возможно, других, подчиненных окон;
• отображение созданного окна и прорисовка содержимого его внутренней части;
• запуск цикла обработки сообщений, помещаемых в очередь приложения;
• завершение работы приложения при извлечении из очереди сообщения WM_QUIT.

Описание параметров функции WinMain()

- hInstance - дескриптор экземпляра приложения. Дескриптор экземпляра приложения - это уникальное число, которое идентифицирует программу, в системе Windows. Каждая копия одной и той же запущенной несколько раз программы называется «экземпляром» и у каждой свое значение hInstance;
- hPrevInstance в настоящее время устарел и всегда равен NULL;
- LpCmdLine - указатель на оканчивающуюся нулем строку, в которой содержатся параметры командной строки;
- nCmdShow - определяет, как приложение первоначально отображается на дисплее: пиктограммой (nCmdShow = SW_SHOWMINNOACTIVE) или в виде открытого окна (nCmdShow = SW_SHOWNORMAL).

Windows-тип данных *дескриптор* (описатель)



используется для описания объектов операционной системы. Дескриптор напоминает индекс хеш-таблицы и позволяет отслеживать состояние объекта в памяти при его перемещении по инициативе операционной системы. Предусмотрено много типов дескрипторов: HINSTANCE, HWND и др., но все они являются 32-разрядными целыми числами.

Внутри головной функции описаны три переменные:

```
4   // Функция WinMain
5   int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
6       LPSTR lpCmdLine, int nCmdShow) {
7       char class_name[] = "main_window"; // Название класса окна
8       char app_title[] = "Simple window"; // Заголовок главного окна
9       WNDCLASS wc; // Структура для информации о классе окна
10      HWND hWnd; // Дескриптор главного окна приложения
11      MSG msg; // Структура для хранения сообщения
```

wc — структура, содержащая информацию по настройке окна;

hWnd — предназначена для хранения дескриптора главного окна программы;

msg — это структура, в которой хранится информация о сообщении, передаваемом операционной системой окну приложения.

wc — структура, содержащая информацию по настройке окна

```
12          // Заполнение структуры WNDCLASS для регистрации класса окна.
13      memset(&wc, 0, sizeof(wc)); // Заполнение структуры нулями
14      wc.lpszClassName = class_name; // Имя класса окон
15      wc.lpfnWndProc = &WndProc; // Адрес оконной функции
16      wc.style = CS_HREDRAW | CS_VREDRAW; // Стиль класса окон
17      wc.hInstance = hInstance; // Экземпляр приложения
18      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // Пиктограмма для окон
19      wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Курсор мыши для окон
20      wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); // Кисть для окон
21      wc.lpszMenuName = NULL; // Ресурс меню окон
22                              // Регистрация класса окна
23      RegisterClass(&wc);
```

wc.hInstance = hInstance; Дескриптор текущего приложения.

wc.style = CS_HREDRAW | CS_VREDRAW;
Такой стиль определяет автоматическую перерисовку окна при изменении его ширины или высоты.

wc — структура, содержащая информацию по настройке окна

```
12          // Заполнение структуры WNDCLASS для регистрации класса окна.
13      memset(&wc, 0, sizeof(wc)); // Заполнение структуры нулями
14      wc.lpszClassName = class_name; // Имя класса окон
15      wc.lpfnWndProc = &WndProc; // Адрес оконной функции
16      wc.style = CS_HREDRAW | CS_VREDRAW; // Стиль класса окон
17      wc.hInstance = 
18      wc.hIcon = LoadIcon(NULL, IDI_APPLICATION); // Пиктограмма для окон
19      wc.hCursor = LoadCursor(NULL, IDC_ARROW); // Курсор мыши для окон
20      wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH); // Кисть для окон
21      wc.lpszMenuName = NULL; // Ресурс меню окон
22                              // Регистрация класса окна
23      RegisterClass(&wc);
```

wc.hIcon = LoadIcon(NULL,IDI_APPLICATION);
Дескриптор пиктограммы (иконки) приложения. Функция LoadIcon() обеспечивает ее загрузку. Если первый параметр NULL, используется системная пиктограмма, которая выбирается по второму параметру из следующего набора:

IDI_ASTERISK — звездочка; IDI_APPLICATION — стандартная иконка;

IDI_EXCLAMATION — восклицательный знак;
IDI_HAND — ладонь;
IDI_QUESTION — вопросительный знак;
IDI_WINLOGO — логотип Windows.

wc — структура, содержащая информацию по настройке окна

wc.hCursor = LoadCursor(NULL,IDC_ARROW);
Аналогичная функция LoadCursor() обеспечивает загрузку графического образа курсора, где нулевой первый параметр также означает использование системного курсора, вид которого можно выбрать из списка:
IDC_ARROW — стандартный курсор;
IDC_APPSTARTING — стандартный курсор и маленькие песочные часы;
IDC_CROSS — перекрестие;
IDC_IBEAM — текстовый курсор;
IDC_NO — перечеркнутый круг;
IDC_SIZEALL — четырехлепестковая стрелка;
IDC_SIZENESW — двухлепестковая стрелка, северо-восток и юго-запад;
IDC_SIZENWSE — двухлепестковая стрелка, северо-запад и юго-восток;
IDC_SIZENS — двухлепестковая стрелка, север и юг;
IDC_SIZEWE — двухлепестковая стрелка, запад и восток;
IDC_UPARROW — стрелка вверх;
IDC_WAIT — песочные часы.

wc — структура, содержащая информацию по настройке окна

lpfnWndProc — (указатель на функцию) указывает адрес оконной функции. Эта функция отвечает за обработку всех сообщений, получаемых окном. Она может обрабатывать эти сообщения сама или вызывать функцию по умолчанию DefWindowProc;

hbrBackground – дескриптор кисти интерфейса GDI, для прорисовки фона окна;

lpszMenuName - определяет ресурс меню (по имени или с помощью макроса MAKEINTRESOURCE по целому идентификатору), который используется для стандартного меню этого класса;

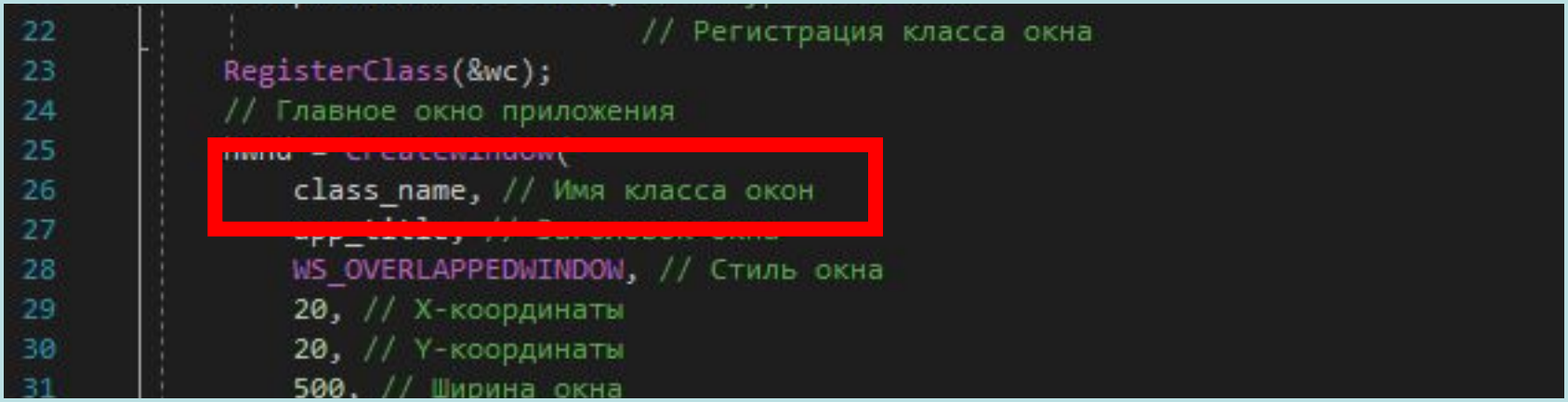
cbClsExtra и cbWndExtra - используются, чтобы выделить дополнительную память для класса окна и для отдельных окон, через них задается размер дополнительной памяти. Приложения могут использовать эту память для хранения специальной информации, относящейся к классу окна или к отдельным окнам.

Создание окна

Регистрация нового класса является первым шагом в создании окна. Затем приложение должно создать окно с помощью функции CreateWindow(), которая возвращает дескриптор созданного окна типа HWND:

```
22                          // Регистрация класса окна
23      RegisterClass(&wc);
24      // Главное окно приложения
25      hWnd = CreateWindow(
26          class_name, // Имя класса окон
27          app_title, // Заголовок окна
28          WS_OVERLAPPEDWINDOW, // Стиль окна
29          20, // X-координаты
30          20, // Y-координаты
31          500, // Ширина окна
32          400, // Высота окна
33          NULL, // Дескриптор родительского окна
34          NULL, // Дескриптор меню окна
35          hInstance, // Дескриптор экземпляра приложения
36          NULL); // Дополнительная информация
37      if (!hWnd) {
38          // Окно не создано, отобразить предупреждение
39          MessageBox(NULL, "Create: error", app_title, MB_OK | MB_ICONSTOP);
40          return FALSE;
```

Описание параметров функции CreateWindow()



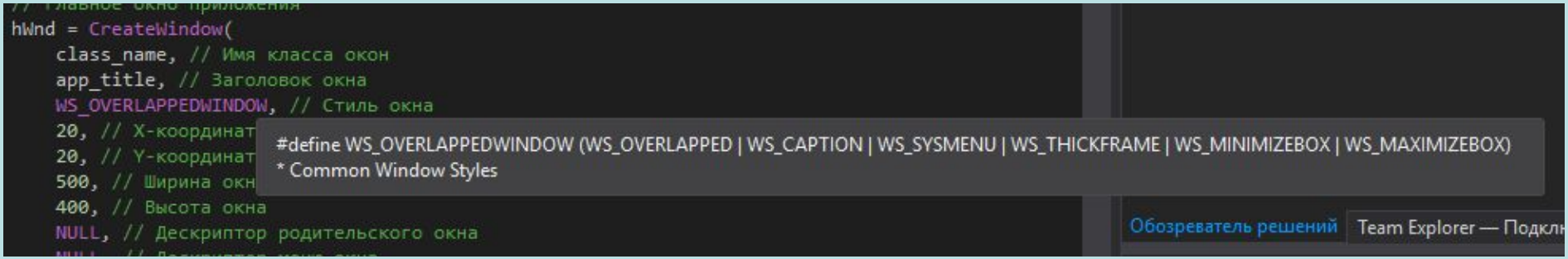
lpClassName - указывает имя класса, поведение которого наследует данное окно. Этот класс должен быть зарегистрирован с помощью фунцкии RegisterClass() или быть одним из предопределенных классов элементов управления. Эти переопределенные классы включают в себя стандартные классы элементов управления с именами “button”, “combobox”, “listbox”, “edit”, “scrollbar” и т. д.

Описание параметров функции CreateWindow()

```
26          class_name, // Имя класса окон
27          app_title, // Заголовок окна
28          WS_OVERLAPPEDWINDOW, // Стиль окна
29          20, // X-координаты
30          20, // Y-координаты
31          500, // Ширина окна
32          400, // Высота окна
```

• lpWindowName - определяет строку, которая выводится в заголовке окна;
• dwStyle - определяет стиль окна. Используется для инициализации локальных свойств окна. Как и в случае стиля класса, стиль окна также обычно является комбинацией значений (объединенных операцией поразрядного ИЛИ). Для определения стиля главного окна чаще всего используют стиль перекрывающегося окна, для чего через параметр dwStyle передают символическую константу WS_OVERLAPPEDWINDOW, определенную во включаемых файлах следующим образом:

Описание параметров функции CreateWindow()



WS_OVERLAPPEDWINDOW — макрос, определяющий стиль отображения стандартного окна, имеющего системное меню, заголовок, рамку для изменения размеров, а также кнопки минимизации, развертывания и закрытия. Это наиболее общий стиль окна.

Описание параметров функции CreateWindow()

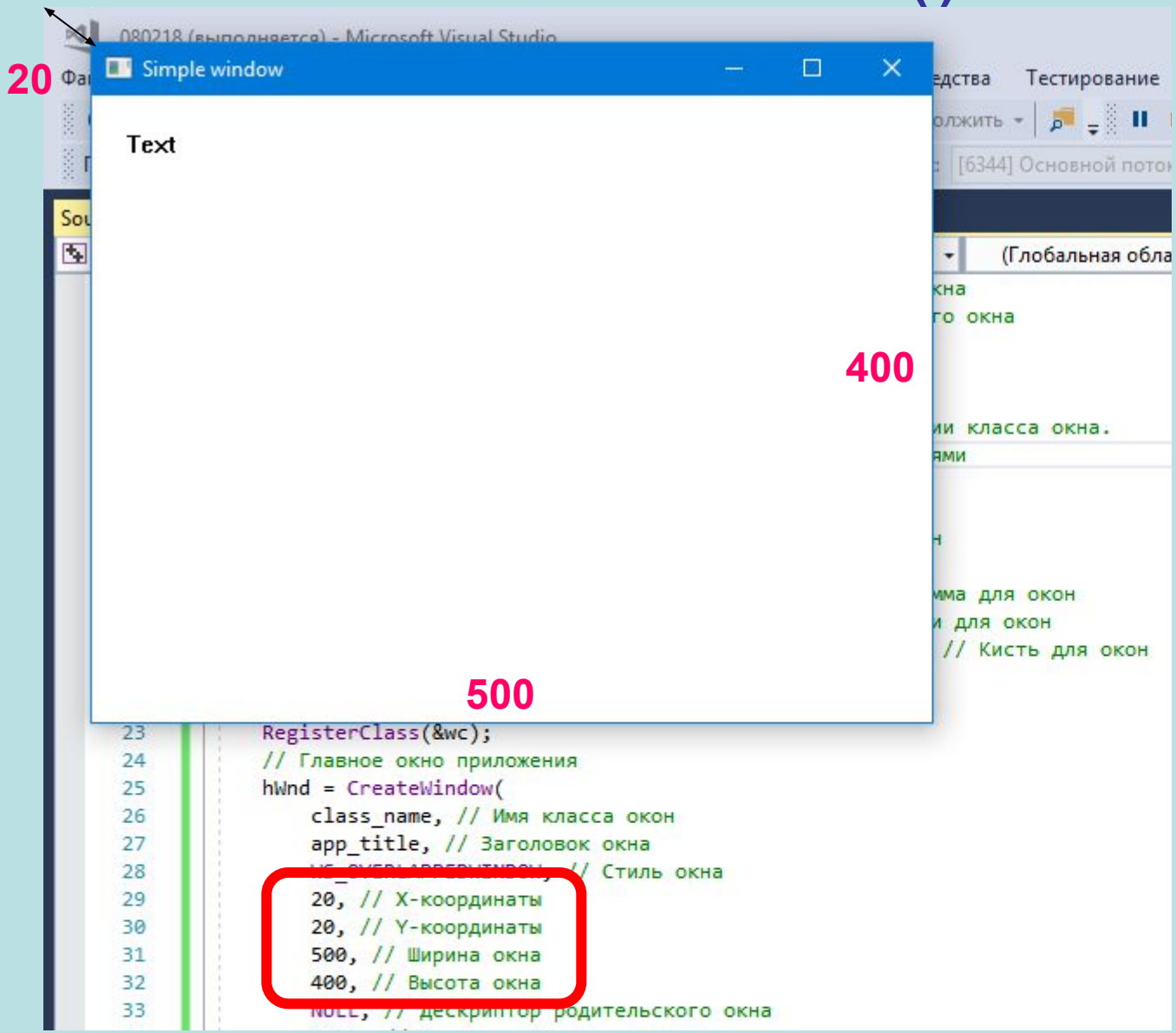
Можно создать другой стиль, используя комбинацию стилевых макросов при помощи операции логического сложения, вот некоторые из них:
WS_OVERLAPPED — стандартное окно с рамкой;
WS_CAPTION — окно с заголовком;
WS_THICKFRAME — окно с рамкой;
WS_MAXIMIZEBOX — кнопка распахивания окна;
WS_MINIMIZEBOX — кнопка минимизации;
WS_SYSMENU — системное меню;
WS_HSCROLL — горизонтальная панель прокрутки;
WS_VSCROLL — вертикальная панель прокрутки;
WS_VISIBLE — окно отображается;
WS_CHILD — дочернее окно;
WS_POPUP — всплывающее окно;

Описание параметров функции CreateWindow()

Следующие два параметра определяют координаты левого верхнего угла окна (x,y), еще два параметра: Width — ширину и Height — высоту окна в пикселах. Задание параметра CW_USEDEFAULT означает, что система сама выберет для отображения окна наиболее (с ее точки зрения) удобное место и размер.

```
26          class_name, // Имя класса окон
27          app_title, // Заголовок окна
28          WS_OVERLAPPEDWINDOW, // Стиль окна
29          20, // X-координаты
30          20, // Y-координаты
31          500, // Ширина окна
32          400, // Высота окна
```

Описание параметров функции CreateWindow()



Описание параметров функции CreateWindow()

```
25      hWnd = CreateWindow(
26          class_name, // Имя класса окон
27          app_title, // Заголовок окна
28          WS_OVERLAPPEDWINDOW, // Стиль окна
29          20, // X-координаты
30          20, // Y-координаты
31          500, // Ширина окна
32          400, // Высота окна
33          NULL, // Дескриптор родительского окна
34          NULL, // Дескриптор меню окна
35          hInstance, // Дескриптор экземпляра приложения
36          NULL); // Дополнительная информация
```

hWndParent — дескриптор родительского окна. Если окно является главным окном приложения, то параметру hWndParent присваивается значение NULL;

• hMenu - дескриптор родительского меню. Значение NULL на месте дескриптора меню hMenu говорит о том, что у окна будет только меню класса, общее для всех окон этого класса;

• hInstance - экземпляр приложения, которое создает окно;

• lpParam - используется для передачи окну дополнительных данных (если их нет, то он должен быть равен NULL).

Отображение окна

Несмотря на то, что функция CreateWindow() создает окно, это не значит, что оно будет автоматически отображаться на экране дисплея. Для отображения окна следует воспользоваться функцией ShowWindow(). Первым параметром в эту функцию передается дескриптор окна, вторым параметром является константа, которая задает начальный вид окна на экране.

```
41      }
42      // Отображение окна.
43      ShowWindow(hWnd, SW_SHOWNORMAL);
44      // Обновление клиентской области окна.
45      UpdateWindow(hWnd);
46      // Запуск цикла обработки очереди сообщений.
```

Функция ShowWindow() выводит окно на экран. Если второй параметр этой функции имеет значение SW_SHOWNORMAL, то фон рабочей области окна закрашивается той кистью, которая задана в классе окна. Если второй параметр имеет значение SW_HIDE — окно будет невидимым.
Функция UpdateWindow() передает функции окна сообщение WM_PAINT. Получив это сообщение, функция обновляет содержимое экрана.

Цикл обработки очереди сообщений

После создания и отображения окна функция WinMain должна подготовить приложение к получению событий от клавиатуры, мыши и т. д. Windows предоставляет очередь сообщений для каждой программы, работающей в данный момент в системе. Когда происходит ввод информации, Windows преобразовывает ее в сообщение, которое помещается в очередь сообщений приложения. Программа извлекает сообщения из очереди, выполняя блок операторов, известный как цикл обработки сообщений (message loop).

```
45      UpdateWindow(hWnd);
46      // Запуск цикла обработки очереди сообщений.
47      // Функция GetMessage получает сообщение из очереди, выдает false при
48      // выборке из очереди сообщения WM_QUIT
49      while (GetMessage(&msg, NULL, 0, 0)) {
50          // Преобразование некоторых сообщений, полученных с помощью клавиатуры
51          TranslateMessage(&msg);
52          // Отправить сообщение оконной функции
53          DispatchMessage(&msg);
54      }
55      return msg.wParam;
56  }
```

Цикл обработки очереди сообщений

Он задается оператором while, аргументом которого является функция GetMessage(&msg, NULL, 0, 0). Такой цикл является обязательным для всех Windows-приложений, его цель — получение и обработка сообщений, передаваемых операционной системой. Операционная система ставит сообщения в очередь, откуда они извлекаются функцией GetMessage() по мере готовности приложения:

первым параметром функции является &msg — указатель на структуру MSG, где и хранятся сообщения;

второй параметр hWnd — определяет окно, для которого предназначено сообщение, если же необходимо перехватить сообщения всех окон данного приложения, он должен быть NULL;

остальные два параметра определяют [min, max] диапазон получаемых сообщений. Чаще всего необходимо обработать все сообщения, тогда эти параметры должны быть равны 0.

Цикл обработки очереди сообщений

```
45      UpdateWindow(hWnd);
46      // Запуск цикла обработки очереди сообщений.
47      // Функция GetMessage получает сообщение из очереди, выдает false при
48      // выборке из очереди сообщения WM_QUIT
49      while (GetMessage(&msg, NULL, 0, 0)) {
50          // Преобразование некоторых сообщений, полученных с помощью клавиатуры
51          TranslateMessage(&msg);
52          // Отправить сообщение оконной функции
53          DispatchMessage(&msg);
54      }
55      return msg.wParam;
56  }
```

Внутри цикла расположены две функции:
TranslateMessage(&msg);
DispatchMessage(&msg);
Первая из них транслирует код нажатой клавиши в клавиатурные сообщения WM_CHAR. При этом в переменную wParam структуры msg помещается код нажатой клавиши в Windows-кодировке CP-1251, в младшее слово lParam — количество повторений этого сообщения в результате удержания клавиши в нажатом состоянии, а в старшее слово — битовая карта клавиатуры.

Цикл обработки очереди сообщений

Использование этой функции не обязательно и нужно только для обработки сообщений от клавиатуры.

Таблица 1.1. *Битовая карта клавиатуры,* `HIWORD(lParam)`

Бит	Значение
15	1, если клавиша отпущена, 0 — если нажата
14	1, если клавиша была нажата перед посылкой сообщения
13	1, если нажата клавиша <Alt>
12—9	Резерв
8	1, если нажата функциональная клавиша
7—0	Scan-код клавиши

Сообщения определяются их номерами, символические имена для них определены в файле включений winuser.h. Префикс всех системных сообщений WM_.

Цикл обработки очереди сообщений

```
45      UpdateWindow(hWnd);
46      // Запуск цикла обработки очереди сообщений.
47      // Функция GetMessage получает сообщение из очереди, выдает false при
48      // выборке из очереди сообщения WM_QUIT
49      while (GetMessage(&msg, NULL, 0, 0)) {
50          // Преобразование некоторых сообщений, полученных с помощью клавиатуры
51          TranslateMessage(&msg);
52          // Отправить сообщение оконной функции
53          DispatchMessage(&msg);
54      }
55      return msg.wParam;
56  }
```

Вторая функция, DispatchMessage(&msg), обеспечивает возврат преобразованного сообщения обратно операционной системе и инициирует вызов оконной функции данного приложения для его обработки.

Данным циклом и заканчивается головная функция.

Оконная функция WndProc()

```
56  }
57  // Оконная функция
58  LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {
59      char str[] = "Text";
60      switch (msg) {
61          // Необходимо обновить содержимое клиентской области окна
62      case WM_PAINT:
63          HDC hDC;
64          PAINTSTRUCT ps;
65          hDC = BeginPaint(hWnd, &ps); // Получить контекст окна
66          TextOut(hDC, 20, 20, str, strlen(str)); // Нарисовать текст
67          EndPaint(hWnd, &ps); // Освободить контекст окна
68          break;
69          // Нажата левая клавиша мыши в клиентской области окна
70      case WM_LBUTTONDOWN:
71          MessageBox(NULL, "Mouse click", "Message", MB_OK | MB_ICONINFORMATION);
72          break;
73          // Закрытие окна.
74      case WM_DESTROY:
75          // В очередь сообщений приложения помещается сообщение WM_QUIT
76          PostQuitMessage(0);
77          break;
78          // Необработанные сообщения передаются в стандартную
79          // функцию обработки сообщений по умолчанию
80      default: return DefWindowProc(hWnd, msg, wParam, lParam);
81      }
82      return 0;
83  }
```

Оконная функция WndProc()

Основной компонент этой функции — переключатель switch, обеспечивающий выбор соответствующего обработчика сообщений по его номеру message. В нашем случае мы предусмотрели обработку лишь одного сообщения WM_DESTROY. Это сообщение посылается, когда пользователь завершает программу.

Получив его, оконная функция вызывает функцию PostQuitMessage(0), которая завершает приложение и передает операционной системе код возврата — 0. Если говорить точнее, генерируется сообщение WM_QUIT, получив которое функция GetMessage() возвращает нулевое значение. В результате цикл обработки сообщений прекращается и происходит завершение работы приложения.
Все остальные сообщения обрабатываются по умолчанию функцией DefWindowProc(), имеющей такой же список параметров и аналогичное возвращаемое значение, поэтому ее вызов помещается после оператора return.

Оконная функция WndProc()

Все четыре параметра оконной функции идентичны первым четырем полям структуры MSG. Первым параметром является дескриптор окна, получающего сообщение. Если в программе создается несколько окон на основе одного и того же класса окна тогда параметр hWnd идентифицирует конкретное окно, которое получает сообщение.

Функция вызывается непосредственно операционной системой Windows и не может вызываться приложением напрямую. Каждое получаемое окном сообщение идентифицируется номером, который содержится в параметре msg оконной функции. Если оконная функция обрабатывает сообщение, то ее возвращаемым значением должен быть 0.

Все сообщения, не обрабатываемые оконной функцией, должны передаваться в функцию DefWindowProc() (такой механизм позволяет Windows обрабатывать окно совместно с приложением). При этом значение, возвращаемое функцией DefWindowProc(), должно быть возвращаемым значением оконной функции. Функция DefWindowProc() играет ключевую роль в формировании информационных потоков сообщений Windows, и ее вызов в функции окна обязателен.

Удаление окна

```
73          // Закрытие окна.
74      case WM_DESTROY:
75          // В очередь сообщений приложения помещается сообщение WM_QUIT
76          PostQuitMessage(0);
77          break;
78          // Необработанные сообщения передаются в стандартную
79          // функцию обработки сообщений по умолчанию
80      default: return DefWindowProc(hWnd, msg, wParam, lParam);
81      }
82      return 0;
83  }
```

Сообщение WM_DESTROY является еще одним важным сообщением. Это сообщение показывает, что Windows находится в процессе ликвидации окна в ответ на полученную от пользователя команду (пользователь вызывает поступление этого сообщения, если нажмет мышью на пиктограмме “Close”, выберет пункт “Close” из системного меню или нажмет комбинацию клавиш Alt+F4).

Главное окно стандартно реагирует на это сообщение, вызывая функцию PostQuitMessage(). Эта функция ставит сообщение WM_QUIT в очередь сообщений приложения. Это заставляет функцию WinMain прервать цикл обработки сообщений и выйти в систему, завершив работу приложения.