Пошаговое руководство. Создание классических приложений Windows (C++)

Visual Studio 2015
Другие версии

Самая актуальная документация по Visual Studio 2017: Документация по Visual Studio 2017.

В этом пошаговом руководстве демонстрируется создание простейшего классического приложения Windows, выводящего в окне надпись "Hello, World!". Код, созданный в этом пошаговом руководстве, можно использовать в качестве шаблона для создания других классических приложений Windows.

API-интерфейс Win32 (также известный как Windows API) — это платформа на основе C для создания приложений Windows. Дополнительные сведения об API-интерфейсе Win32 см. в разделе Windows API.

Важно

Для того чтобы более понятно объяснить определенные сегменты кода, используемые в этом пошаговом руководстве, мы опустили некоторые операторы кода, необходимые в реально работающем приложении, например директивы включения и объявления глобальных переменных. В разделе **Пример** в конце этого документа показан полный код.

Обязательные компоненты

Для выполнения этого пошагового руководства читатель должен владеть основами языка C++.

Видеодемонстрация доступна в разделе Видео. Практическое руководство. Создание приложений Win32 (C++) документации по Visual Studio 2008.

Создание проекта на основе Win32

1. В меню **Файл** последовательно выберите пункты **Создать** и **Проект**.
2. В левой области диалогового окна **Новый проект** щелкните **Установленные шаблоны**, выберите **Visual C++** и щелкните **Win32**. В средней области выберите шаблон **Проект Win32**.

   В поле **Имя** введите имя проекта, например `win32app`. Нажмите кнопку **ОК**.

3. На начальной странице **мастера приложений Win32** нажмите кнопку **Далее**.
4. На странице "Параметры приложения" в разделе **Тип приложения** выберите **Приложение Windows**. В поле **Дополнительные параметры** выберите **Пустой проект**. Чтобы создать проект, нажмите кнопку **Готово**.

5. В **обозревателе решений** щелкните правой кнопкой мыши проект Win32app, выберите пункт **Добавить**, а затем пункт **Новый элемент**. В диалоговом окне **Добавление нового элемента** выберите **Файл C++ (.cpp)**. В поле **Имя** введите имя файла, например `GT_HelloWorldWin32.cpp`. Нажмите кнопку **Добавить**.

Запуск классического приложения Windows

1. Точно так же, как каждое приложение на языке C и C++ должно иметь в качестве начальной точки функцию `main`, каждое приложение на основе Win32 должно иметь функцию `WinMain`.`WinMain` имеет следующий синтаксис:

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

Сведения о параметрах и значениях, возвращаемых этой функцией, см. в разделе Функция WinMain.

4. Так как в коде приложения должны использоваться существующие определения, следует добавить в файл операторы включения.

```
#include <windows.h> #include <stdlib.h> #include <string.h> #include <tchar.h>
```

7. Наряду с функцией `WinMain` в каждом классическом приложении Windows также должна быть определена функция оконной процедуры. Обычно эта функция имеет имя `WndProc`.`WndProc` имеет следующий синтаксис:

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

Эта функция обрабатывает многочисленные *сообщения*, которые приложение получает от операционной системы. Например, в приложении с диалоговым окном, в котором есть кнопка **ОК**, при нажатии упомянутой пользователем операционная система отправляет в приложение сообщение о том, что эта кнопка была нажата. Функция `WndProc` отвечает за реагирование на это событие. В этом примере соответствующей реакцией на это событие может быть закрытие диалогового окна.

Дополнительные сведения см. в разделе Процедуры окна.

Добавление функциональных возможностей в функцию WinMain

1. В функции `WinMain` создайте структуру класса окна типа WNDCLASSEX. Эта структура содержит информацию об окне, такую как используемые в приложении значки, цвет фона окна, отображаемое в заголовке окна название, имя функции процедуры окна и т. д. В приведенном ниже примере показана типичная структура `WNDCLASSEX`.

```
WNDCLASSEX wcex; wcex.cbSize = sizeof(WNDCLASSEX); wcex.style = CS_HREDRAW | CS_VREDRAW; wcex.lpfnWndProc    = WndProc; wcex.cbClsExtra     = 0; wcex.cbWndExtra     = 0; wcex.hInstance      = hInstance; wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION)); wcex.hCursor        = LoadCursor(NULL, IDC_ARROW); wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1); wcex.lpszMenuName   = NULL; wcex.lpszClassName  = szWindowClass; wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
```

Сведения о полях этой структуры см. в разделе WNDCLASSEX.

4. После того как класс окна создан, необходимо зарегистрировать его. Воспользуйтесь функцией RegisterClassEx и передайте структуру класса окна в качестве аргумента.

```
5. if (!RegisterClassEx(&wcex)) { MessageBox(NULL, _T("Call to
   RegisterClassEx failed!"), _T("Win32 Guided Tour"), NULL); return 1; }
6.
```

7. Теперь можно создать окно. Воспользуйтесь функцией CreateWindow.

```
8. static TCHAR szWindowClass[] = _T("win32app"); static TCHAR szTitle[] =
   _T("Win32 Guided Tour Application"); // The parameters to CreateWindow
   explained: // szWindowClass: the name of the application // szTitle:
   the text that appears in the title bar // WS_OVERLAPPEDWINDOW: the type
   of window to create // CW_USEDEFAULT, CW_USEDEFAULT: initial position
   (x, y) // 500, 100: initial size (width, length) // NULL: the parent of
   this window // NULL: this application does not have a menu bar //
   hInstance: the first parameter from WinMain // NULL: not used in this
   application HWND hWnd = CreateWindow( szWindowClass, szTitle,
   WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 500, 100, NULL,
   NULL, hInstance, NULL ); if (!hWnd) { MessageBox(NULL, _T("Call to
   CreateWindow failed!"), _T("Win32 Guided Tour"), NULL); return 1; }
9.
```

Эта функция возвращает объект HWND, являющийся дескриптором окна. Дополнительные сведения см. в разделе Типы данных Windows.

10. Теперь воспользуйтесь приведенным ниже кодом, чтобы отобразить окно.

```
11.  // The parameters to ShowWindow explained: // hWnd: the value returned
   from CreateWindow // nCmdShow: the fourth parameter from WinMain
   ShowWindow(hWnd, nCmdShow); UpdateWindow(hWnd);
12.
```

На этом этапе в окне не будет отображаться большое количество содержимого, так как функция `WndProc` еще не реализована.

13. Теперь добавьте цикл обработки сообщений для прослушивания отправляемых ОС сообщений. Когда приложение получает сообщение, этот цикл пересылает его функции `WndProc` для обработки. Цикл обработки сообщений напоминает приведенный ниже код.

```
14.  MSG msg; while (GetMessage(&msg, NULL, 0, 0)) {
   TranslateMessage(&msg); DispatchMessage(&msg); } return (int)
   msg.wParam;
15.
```

Дополнительные сведения о структурах и функциях, используемых в цикле обработки сообщений, см. в разделах, посвященных MSG, GetMessage, TranslateMessage и DispatchMessage.

На этом этапе функция `WinMain` должна напоминать приведенный ниже код.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) { WNDCLASSEX wcex; wcex.cbSize =
sizeof(WNDCLASSEX); wcex.style          = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc    = WndProc; wcex.cbClsExtra     = 0; wcex.cbWndExtra
= 0; wcex.hInstance      = hInstance; wcex.hIcon          =
LoadIcon(hInstance, MAKEINTRESOURCE(IDI_APPLICATION)); wcex.hCursor
= LoadCursor(NULL, IDC_ARROW); wcex.hbrBackground  =
(HBRUSH)(COLOR_WINDOW+1); wcex.lpszMenuName   = NULL;
```

```
wcex.lpszClassName  = szWindowClass; wcex.hIconSm        =
LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION)); if
(!RegisterClassEx(&wcex)) { MessageBox(NULL, _T("Call to
RegisterClassEx failed!"), _T("Win32 Guided Tour"), NULL); return 1; }
hInst = hInstance; // Store instance handle in our global variable //
The parameters to CreateWindow explained: // szWindowClass: the name of
the application // szTitle: the text that appears in the title bar //
WS_OVERLAPPEDWINDOW: the type of window to create // CW_USEDEFAULT,
CW_USEDEFAULT: initial position (x, y) // 500, 100: initial size
(width, length) // NULL: the parent of this window // NULL: this
application dows not have a menu bar // hInstance: the first parameter
from WinMain // NULL: not used in this application HWND hWnd =
CreateWindow( szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT, 500, 100, NULL, NULL, hInstance, NULL );
if (!hWnd) { MessageBox(NULL, _T("Call to CreateWindow failed!"),
_T("Win32 Guided Tour"), NULL); return 1; } // The parameters to
ShowWindow explained: // hWnd: the value returned from CreateWindow //
nCmdShow: the fourth parameter from WinMain ShowWindow(hWnd, nCmdShow);
UpdateWindow(hWnd); // Main message loop: MSG msg; while
(GetMessage(&msg, NULL, 0, 0)) { TranslateMessage(&msg);
DispatchMessage(&msg); } return (int) msg.wParam; }
```

Добавление функциональных возможностей в функцию WndProc

1. Чтобы включить обработку получаемых приложением сообщений функцией `WndProc`, реализуйте оператор switch.

   Первым обрабатывается сообщение WM_PAINT. Приложение получает это сообщение, когда часть его отображаемого окна требует обновления. (При первом отображении окна его требуется обновить полностью.)

   Для обработки сообщения `WM_PAINT` сначала вызовите метод BeginPaint, далее обработайте логику расположения текста, кнопок и других элементов управления в окне, а затем вызовите метод EndPaint. В этом приложении логика между начальным и конечным вызовами предполагает отображение в окне строки "Hello, World!". В приведенном ниже коде обратите внимание, что функция TextOut используется для отображения строки.

   ```
   PAINTSTRUCT ps; HDC hdc; TCHAR greeting[] = _T("Hello, World!"); switch
   (message) { case WM_PAINT: hdc = BeginPaint(hWnd, &ps); // Here your
   application is laid out. // For this introduction, we just print out
   "Hello, World!" // in the top left corner. TextOut(hdc, 5, 5, greeting,
   _tcslen(greeting)); // End application-specific layout section.
   EndPaint(hWnd, &ps); break; }
   ```

2. Обычно приложение обрабатывает много других сообщений, например WM_CREATE и WM_DESTROY. В приведенном ниже коде содержится базовое представление полной функции `WndProc`.

3. ```
   LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
   lParam) { PAINTSTRUCT ps; HDC hdc; TCHAR greeting[] = _T("Hello,
   World!"); switch (message) { case WM_PAINT: hdc = BeginPaint(hWnd,
   &ps); // Here your application is laid out. // For this introduction,
   we just print out "Hello, World!" // in the top left corner.
   TextOut(hdc, 5, 5, greeting, _tcslen(greeting)); // End application
   specific layout section. EndPaint(hWnd, &ps); break; case WM_DESTROY:
   PostQuitMessage(0); break; default: return DefWindowProc(hWnd, message,
   wParam, lParam); break; } return 0; }
   ```
   ```

4.

Пример

Сборка примера

1. Создайте проект на основе Win32, как показано в разделе "Создание проекта на основе Win32" ранее в этом пошаговом руководстве.
2. Скопируйте код, позволяющий выполнить эти шаги, и вставьте его в исходный файл GT_HelloWorldWin32.cpp.
3. В меню **Сборка** выберите **Собрать решение**.
4. Чтобы запустить приложение, нажмите клавишу F5. Окно, содержащее текст "Hello, World!", должно отображаться в левом верхнем углу экрана.

Код

```
// GT_HelloWorldWin32.cpp // compile with: /D_UNICODE /DUNICODE /DWIN32
/D_WINDOWS /c #include <windows.h> #include <stdlib.h> #include <string.h>
#include <tchar.h> // Global variables // The main window class name. static
TCHAR szWindowClass[] = _T("win32app"); // The string that appears in the
application's title bar. static TCHAR szTitle[] = _T("Win32 Guided Tour
Application"); HINSTANCE hInst; // Forward declarations of functions included
in this code module: LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nCmdShow) { WNDCLASSEX wcex; wcex.cbSize = sizeof(WNDCLASSEX);
wcex.style          = CS_HREDRAW | CS_VREDRAW; wcex.lpfnWndProc    = WndProc;
wcex.cbClsExtra     = 0; wcex.cbWndExtra     = 0; wcex.hInstance      =
hInstance; wcex.hIcon          = LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_APPLICATION)); wcex.hCursor        = LoadCursor(NULL,
IDC_ARROW); wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1); wcex.lpszMenuName
= NULL; wcex.lpszClassName  = szWindowClass; wcex.hIconSm        =
LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_APPLICATION)); if
(!RegisterClassEx(&wcex)) { MessageBox(NULL, _T("Call to RegisterClassEx
failed!"), _T("Win32 Guided Tour"), NULL); return 1; } hInst = hInstance; //
Store instance handle in our global variable // The parameters to
CreateWindow explained: // szWindowClass: the name of the application //
szTitle: the text that appears in the title bar // WS_OVERLAPPEDWINDOW: the
type of window to create // CW_USEDEFAULT, CW_USEDEFAULT: initial position
(x, y) // 500, 100: initial size (width, length) // NULL: the parent of this
window // NULL: this application does not have a menu bar // hInstance: the
first parameter from WinMain // NULL: not used in this application HWND hWnd
= CreateWindow( szWindowClass, szTitle, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
CW_USEDEFAULT, 500, 100, NULL, NULL, hInstance, NULL ); if (!hWnd) {
MessageBox(NULL, _T("Call to CreateWindow failed!"), _T("Win32 Guided Tour"),
NULL); return 1; } // The parameters to ShowWindow explained: // hWnd: the
value returned from CreateWindow // nCmdShow: the fourth parameter from
WinMain ShowWindow(hWnd, nCmdShow); UpdateWindow(hWnd); // Main message loop:
MSG msg; while (GetMessage(&msg, NULL, 0, 0)) { TranslateMessage(&msg);
DispatchMessage(&msg); } return (int) msg.wParam; } // //  FUNCTION:
WndProc(HWND, UINT, WPARAM, LPARAM) // //  PURPOSE:  Processes messages for
the main window. // //  WM_PAINT    - Paint the main window //  WM_DESTROY  -
post a quit message and return // // LRESULT CALLBACK WndProc(HWND hWnd, UINT
message, WPARAM wParam, LPARAM lParam) { PAINTSTRUCT ps; HDC hdc; TCHAR
greeting[] = _T("Hello, World!"); switch (message) { case WM_PAINT: hdc =
BeginPaint(hWnd, &ps); // Here your application is laid out. // For this
introduction, we just print out "Hello, World!" // in the top left corner.
TextOut(hdc, 5, 5, greeting, _tcslen(greeting)); // End application-specific
layout section. EndPaint(hWnd, &ps); break; case WM_DESTROY:
PostQuitMessage(0); break; default: return DefWindowProc(hWnd, message,
wParam, lParam); break; } return 0; }
```

https://msdn.microsoft.com/ru-ru/library/bb384843.aspx

См. также

Классические приложения Windows

Практическое руководство. Создание классического приложения Windows

Visual Studio 2015
Другие версии

Создавать приложение Windows Win32 проще всего с помощью мастера приложений Win32.

Чтобы создать приложение Windows Win32, выполните следующие действия:

1. Следуйте инструкциям, приведенным в разделе справки Создание проекта с использованием мастера приложений Visual C++.
2. В левой области диалогового окна **Новый проект** разверните узел **Visual C++** и выберите **Win32**. Далее выделите **Проект Win32** в средней области, введите имя проекта и щелкните "ОК", чтобы открыть мастер.
3. Определите параметры приложения с помощью мастера приложений Win32.

   Примечание

   Для сохранения параметров, заданных в мастере по умолчанию, пропустите этот шаг.

4. Нажмите кнопку **Готово**, чтобы закрыть мастер, и созданный проект откроется в **Обозревателе решений**.

См. также

Добавление функциональных возможностей с помощью мастеров кода
Страницы свойств (Visual C++)
Deploying Applications

Создание пустого классического приложения Windows

Visual Studio 2015
Другие версии

Самая актуальная документация по Visual Studio 2017: Документация по Visual Studio 2017.

Создание пустого классического приложения Windows

1. В меню **Файл** выберите пункт **Создать > Проект**.
2. В левой области диалогового окна **Новый проект** щелкните **Win32**, а в центральной области выберите **Консольное приложение Win32**.
3. Введите имя нового проекта, путь к каталогу проекта, после чего нажмите кнопку **ОК**.
4. В мастере приложений Win32 щелкните страницу **Параметры приложения**. Выберите **тип приложения**, которое нужно создать с помощью файла исходного кода, а затем установите флажок **Пустой проект** в разделе **Дополнительные параметры**.
5. Нажмите кнопку **ОК**.

В **обозревателе решений** появится проект с тремя каталогами, которые должны содержать исходные файлы, файлы заголовков и файлы ресурсов.

Далее можно добавить файлы в пустой проект Visual C++.

См. также

Deploying Applications

Working with Resource Files

Visual Studio 2015
Другие версии

Самая актуальная документация по Visual Studio 2017: Документация по Visual Studio 2017.

Предупреждение

Этот раздел относится к классическим приложениям Windows, написанным на C++.

Сведения о других ресурсах в приложениях Магазин Windows 8.x на C++ см. в разделе Определение ресурсов приложения.

Сведения о добавлении ресурсов в проекты C++ и CLI см. в разделе Ресурсы приложений*Руководства разработчика .NET Framework*.

Ресурсы могут состоять из различных элементов, в том числе элементов интерфейса, которые предоставляют сведения пользователю (например, растровое изображение, значок или указатель), настраиваемых ресурсов, которые содержат данные приложения, ресурсов версии, которые используются API установки, а также ресурсов меню и диалоговых окон.

Вы можете добавить новые ресурсы в проект и изменить их с помощью соответствующего редактора ресурсов. Большинство мастеров Visual C++ автоматически создают RC-файл для проекта.

Сведения о добавлении ресурсов в проекты управляемого кода см. в разделе Ресурсы приложений*Руководства разработчика .NET Framework*. Сведения о том, как вручную добавлять файлы ресурсов в проекты управляемого кода, осуществлять доступ к ресурсам, отображать статические ресурсы и присваивать строки ресурсов свойствам, см. в разделах Пошаговое руководство. Локализация Windows Forms и Walkthrough: Using Resources for Localization with ASP.NET.

В этом подразделе

Файлы ресурсов
В этой статье описываются файлы ресурсов и их использование в классических приложениях Windows. Здесь также представлены ссылки на разделы, посвященные применению файлов ресурсов.

Символы: идентификаторы ресурсов
В этой статье описываются символы и использование диалогового окна **Символы ресурсов** для управления символами в проекте.

Редакторы ресурсов
В этой статье описываются редакторы ресурсов, доступные в Visual Studio, и типы ресурсов, которые можно изменить в каждом редакторе. Также здесь представлены ссылки на подробные сведения о работе с каждым редактором.

Связанные подразделы

Visual C++
Ссылки на документацию по Visual C++.

Введение в Visual Studio
Описание полного набора средств разработки, которые используют одну интегрированную среду разработки (IDE), что позволяет им совместно использовать инструменты и создавать решения на разных языках.

Обращайтесь к нам
Ссылки на сведения об использовании документации, обращении в службу поддержки и использовании специальных возможностей.

См. также

Классические приложения Windows
Меню и другие ресурсы