Взаимодействие Windows API с помощью PowerShell

Боярова Елизавета

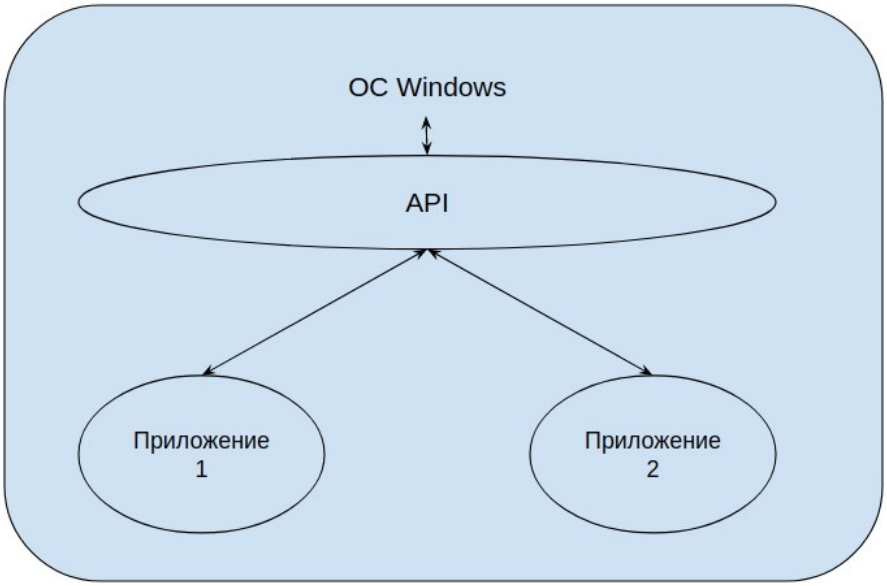
Январь 2024

Содержание

1 ОБЩИЕ СВЕДЕНИЯ

1.1 Microsoft Windows API

Microsoft Windows API — интерфейс системного программирования для операционных систем Windows. Необходим для того, чтобы написанные приложения могли взаимодействовать с операционной системой.



Windows API (Win32) ориентирован главным образом на язык программирования C, поскольку его предоставляемые функции и структуры данных описаны на этом языке в последних версиях его документации. Однако API может использоваться любым компилятором или ассемблером языка программирования, способным обрабатывать (четко определенные) низкоуровневые структуры данных вместе с предписанными соглашениями о вызовах для вызовов и обратных вызовов.

1.2 PowerShell

PowerShell – это стандартизированная оболочка командной строки, которая открывает доступ к более гибкому управлению компьютером и его настройкам. Это та же командная строка, но с большими возможностями для конфигурирования операционных систем семейства MS Windows. Проще говоря, это своего рода универсальный инструмент администрирования. В отличие от большинства оболочек, которые только принимают и возвращают текст, PowerShell принимает и возвращает объекты .NET.

Это решение предлагает следующие возможности:

- надежный журнал командной строки;
- заполнение нажатием клавиши TAB и подстановка команд;
- поддержка псевдонимов команд и параметров;
- создание конвейера для объединения команд;
- система справки консоли, похожая на страницы man в Unix.

1.3 Команды PowerShell

Оболочка PowerShell имеет собственный набор команд, именуемый командлетами (от английского «cmdlet»). Они формируются с помощью шаблона «Глагол-Существительное», или «Действие-Объект».

Например, *Get-Services* и *Start-Process*.

Основные команды языка Windows PowerShell

Ниже таблице перечислены основные команды PowerShell и их аналоги в *nix-подобных системах и CMD.EXE.

Командлет (псевдоним)	Команда в nix	Команда в CMD.exe	Описание
Get-Location (pwd)	pwd		Выводит путь до текущего каталога
Set-Location (cd)	cd	cd	Меняет текущий каталог
Get-ChildItem (ls)	ls	dir	Выводит содержимое текущего каталога
Get-ChildItem	find	find	Производит поиск файлов по заданным критериям
Copy-Item (cp)	cp	cp	Копирует файл
Remove-Item (rm)	rm	rm	Удаляет файл

1.4 Синтаксис

После имени самого командлета следует указание параметров и их значений. Между всеми частями команды следует проставлять пробелы.

Пример: *Set-Location -LiteralPath C: -PassThru*

Разберем пример:

Set-Location - буквально «вызвать команду». Позволяет выполнить указанный блок сценария.

-LiteralPath C: – здесь передаем блок сценария, в котором используется команда Set-Location для перехода в каталог C:˙

-PassThru – по умолчанию командлет Invoke-Command не возвращает результат выполнения. Этот параметр указывает на необходимость вывода информации о местоположении, в которое был выполнен переход с помощью команды Set-Location.

Регистр букв в командах PowerShell не имеет значения, команды будут выполнятся в любом регистре. Когда в одной строке объединены несколько команд, они разделяются точкой с запятой `;` .

Если команда вышла слишком длинной, то для разделения на несколько строк используйте символ обратного апострофа `` ` ``, вместо переноса. Новую строку можно создать, нажав **Shift** + **Enter** (для переноса строки ниже текущей) или **Ctrl** + **Enter** (для переноса строки выше текущей).

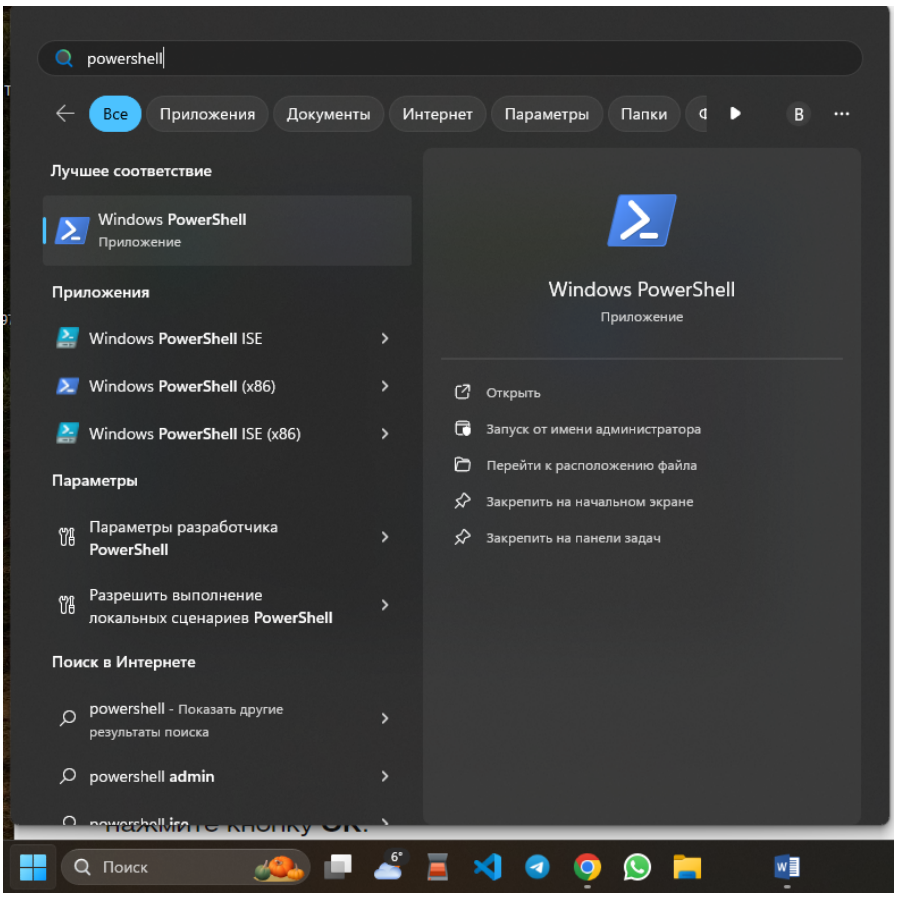
2 НАЧАЛО РАБОТЫ

2.1 Как запустить PowerShell в Windows

По умолчанию PowerShell уже установлен на вашем компьютере. Если нет, то воспользуйтесь инструкциями для установки, предоставленными Microsoft.

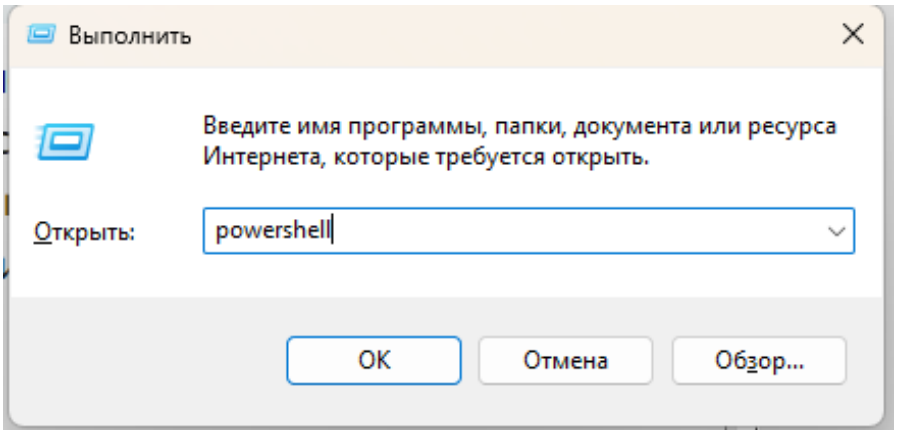
Существуют два основных способа запуска PowerShell в системе Windows: через меню «Пуск» и с помощью приложения «Выполнить».

Через меню «Пуск»:



Через приложение «Выполнить»:

Используйте комбинацию клавиш **Win** + **R**. В появившемся окне введите powershell и нажмите кнопку **OK**.



2.2 Взаимодействие с функциями Windows API с помощью PowerShell

В Windows PowerShell существует три способа взаимодействия с функциями Windows API:

- С помощью командлета $Add-Type$ для компиляции кода C#;
- С помощью ссылки на приватный тип в .NET Framework вызвать данный метод;
- С помощью рефлексии, чтобы определить метод, вызывающий функцию Windows API.

2.3 Использование командлета Add-Type для вызова функции

Ключ к взаимодействию с функциями WinAPI через PowerShell – знать, как преобразовывать типы в C/C++ функциях в их эквиваленты в фреймворке .NET. Для этой цели используется сайт *pinvoke.net*.

На сайте *pinvoke.net* в поисковой строке нужно указать имя искомой функции из WinAPI, например, $CopyFile$. Результатом поиска станет сигнатура эквивалентной C# функции:

$$[DllImport(\text{“}kernel32.dll\text{”}, CharSet = CharSet.Unicode)]$$
$$static\ extern\ bool\ CopyFile(string\ lpExistingFileName,$$
$$string\ lpNewFileName,\ bool\ bFailIfExists);$$

Далее необходимо использовать командлет $Add-Type$ для взаимодействия с найденной C# функцией. Этот командлет позволяет на лету скомпилировать добавленный в него C# код.

Добавим найденную функцию $CopyFile$ в PowerShell скрипт:

```
$MethodDefinition = @'
[DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
public static extern bool CopyFile(string lpExistingFileName, string
lpNewFileName, bool bFailIfExists);
'@
Add-Type -MemberDefinition $MethodDefinition -Name 'Kernel32' -Namespace
'Win32' -PassThru
```

Переменная $MethodDefinition$ содержит сигнатуру функции $CopyFile$, найденную на сайте *pinvoke.net* с одним изменением: функция $CopyFile$ объявлена как *public*, так как функции, который добавляются с помощью $Add-Type$ должны быть публичными для облегчения взаимодействия с PowerShell.

Далее, с помощью вызова Add-Type обеспечивается доступ к C# коду через указание имени типа и пространства имен. После вызова Add-Type вы можете использовать новый тип в PowerShell как [Win32.Kernel32].

Например, теперь можно вызвать функцию $CopyFile$ из PowerShell с помощью команды:

```
[Win32.Kernel32]::CopyFile("C:_1.txt  "C:_2.txt $False)
```

Эта команда скопирует файл **file_1.txt** в файл **file_2.txt**.

Таким образом мы смогли вызвать функцию CopyFile из WinApi в PowerShell.