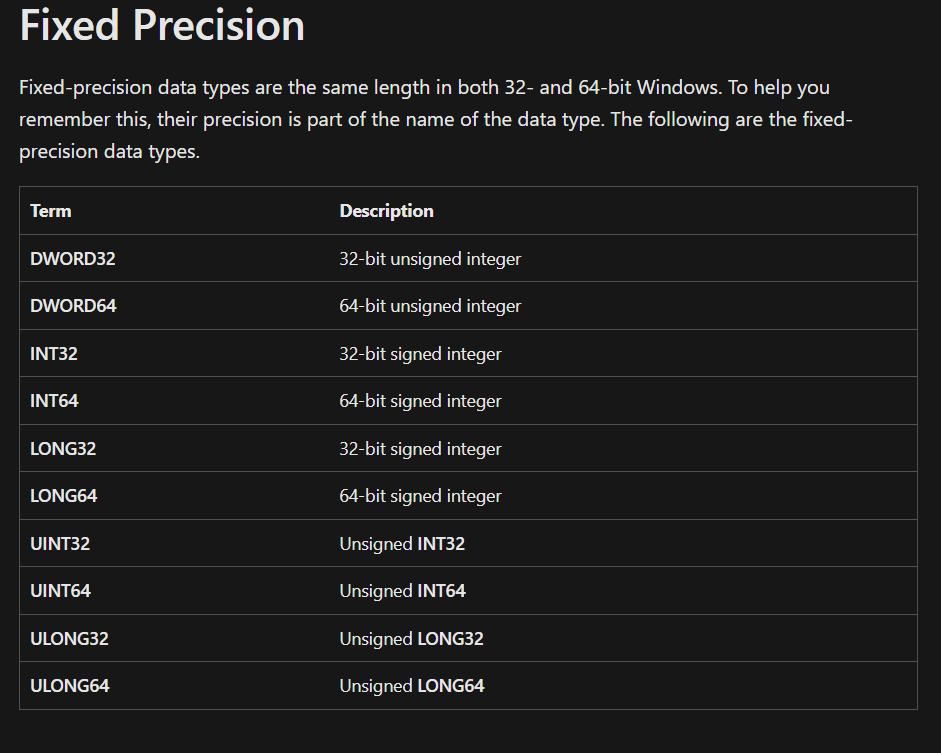
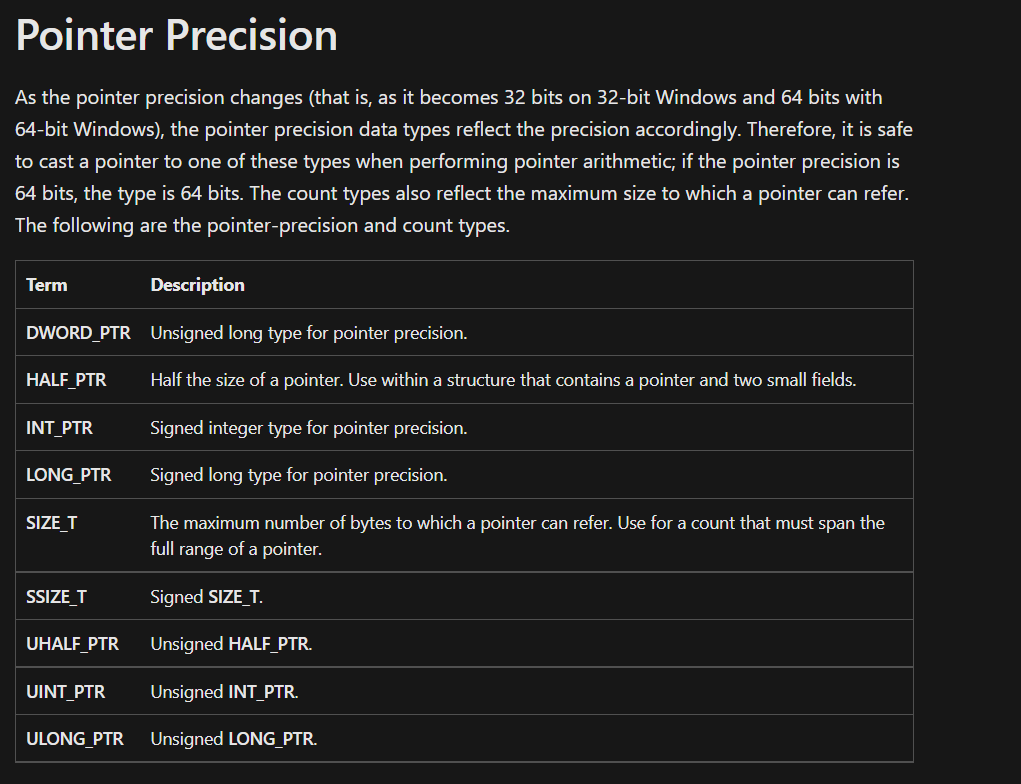
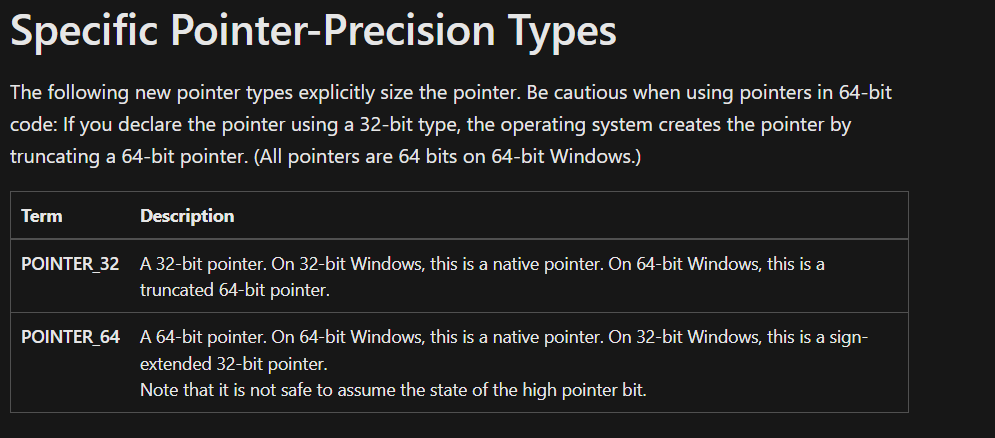
**1.Windows Data Types:**

Three classes of data types were introduced for 64-bit Windows: fixed-precision data types, pointer-precision types, and specific-pointer-precision types.

Header files where data types are defined.

* BaseTsd.h
* WinDef.h
* WinNT.h

**2. Win32 API**

The Win32 API (also called the Windows API) is the original platform for native C/C++ Windows applications that require direct access to Windows and hardware.

The main components of the WinAPI are:

* WinBase: The kernel functions, CreateFile, CreateProcess, etc
* WinUser: The GUI functions, CreateWindow, RegisterClass, etc
* WinGDI: The graphics functions, Ellipse, SelectObject, etc
* Common controls: Standard controls, list views, sliders, etc

#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nCmdShow)

{

MessageBox(NULL, "Hello ,ZOHO!", "Note", MB\_OK);

return 0;

}

WinMain() is windows equivalent of main() from DOS or UNIX. This is where your program starts execution. The parameters are as follows:

**HINSTANCE hInstance**

Handle to the programs executable module (the .exe file in memory)

**HINSTANCE hPrevInstance**

Always NULL for Win32 programs.

**LPSTR lpCmdLine**

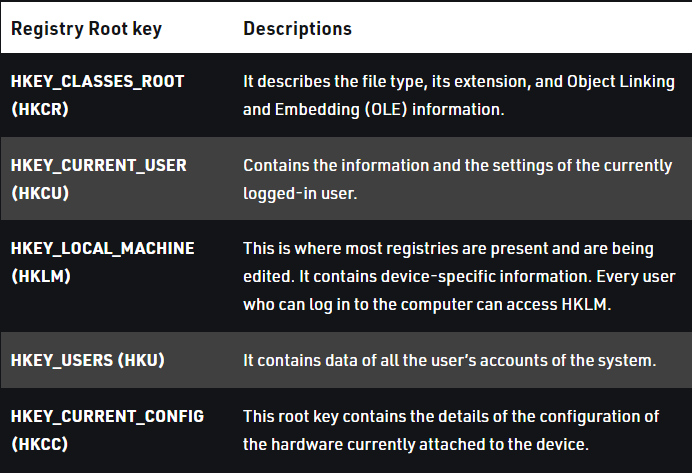
The command line arguments as a single string. NOT including the program name.

**int nCmdShow**

An integer value which may be passed to ShowWindow().

**3.Windows Registry:**

* + The Windows Registry is a collection of databases of configuration settings for Microsoft Windows operating systems.
  + This part of Windows stores much of the information and settings for software programs, hardware devices, user preferences, and operating system configurations.
  + when a new program is installed, a new set of instructions and file references may be added to the registry in a specific location for the program, and others that may interact with it, to refer to for more information like where the files are located, which options to use in the program, etc
  + The registry contains registry values (which are instructions), located within registry keys (folders that contain more data), all within one of several registry hives (folders that categorize all the data in the registry using subfolders).
  + Registry backup files are saved as REG files.
  + The SAM, SECURITY, SOFTWARE, SYSTEM, and DEFAULT registry files, among others, are stored in newer versions of Windows (Windows XP through Windows 11) in this System32\config
  + Older versions of Windows use the %WINDIR% folder to store registry data as DAT files.



**Creating registry key:**

#include <tchar.h>

#include <Windows.h>

#include <iostream>

using namespace std;

BOOL CreateRegistryKey(HKEY hKeyParent,LPCSTR subkey)

{

    DWORD dwDisposition; //It verify new key is created or open existing key

    HKEY  hKey;

    DWORD Ret;

    Ret =

        RegCreateKeyEx(

            hKeyParent,

            subkey,

            0,

            NULL,

            REG\_OPTION\_NON\_VOLATILE,

            KEY\_ALL\_ACCESS | KEY\_WOW64\_64KEY,

            NULL,

            &hKey,

            &dwDisposition);

    if (Ret != ERROR\_SUCCESS)

    {

        printf("Error opening or creating new key\n");

        return FALSE;

    }

    RegCloseKey(hKey); //close the key

    return TRUE;

}

int main()

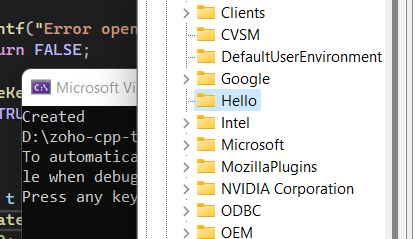
{

    LPCSTR t= \_T("Software\\Hello");

    CreateRegistryKey(HKEY\_LOCAL\_MACHINE,t);

    return 0;

}



Deleting Registy Key:

#include <Windows.h>

#include <iostream>

using namespace std;

BOOL DeleteRegistryKey(HKEY hKeyParent, LPCWSTR subkey) {

    DWORD Ret;

    Ret = RegDeleteKey(hKeyParent, subkey);

    if (Ret != ERROR\_SUCCESS)

    {

        printf("Error opening or creating new key\n");

        return FALSE;

    }

    return TRUE;

}

int main()

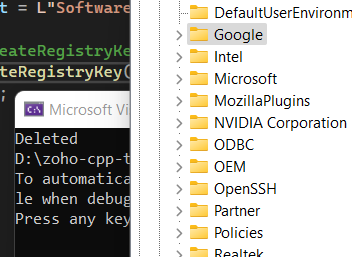
{

    LPCWSTR t = L"Software\\Hello";

    if (DeleteRegistryKey(HKEY\_LOCAL\_MACHINE, t))cout << "Deleted";

    return 0;

}



**4.Windows Process:**

* A process contains its own independent virtual address space with both code and data, protected from other processes. Each process, in turn, contains one or more independently executing threads. A thread running within a process can execute application code, create new threads, create new independent processes, and manage communication and synchronization among the threads.
* Every process contains one or more threads, and the Windows thread is the basic executable unit
* Threads are scheduled on the basis of the usual factors: availability of resources such as CPUs and physical memory, priority, fairness, and so on.
* Windows has long supported multiprocessor systems, so threads can be allocated to separate processors within a computer.
* Each Windows process includes resources such as the following components:
* One or more threads.
* A virtual address space that is distinct from other processes' address spaces. Note that shared memory-mapped files share physical memory, but the sharing processes will probably use different virtual addresses to access the mapped file.
* One or more code segments, including code in DLLs.
* One or more data segments containing global variables.
* Environment strings with environment variable information, such as the current search path.
* The process heap.
* Resources such as open handles and other heaps.
* Each thread in a process shares code, global variables, environment strings, and resources. Each thread is independently scheduled, and a thread has the following elements:
* A stack for procedure calls, interrupts, exception handlers, and automatic storage.
* Thread Local Storage (TLS)—An arraylike collection of pointers giving each thread the ability to allocate storage to create its own unique data environment.
* An argument on the stack, from the creating thread, which is usually unique for each thread.
* A context structure, maintained by the kernel, with machine register values.

**5.Windows Services:**

* Windows Services are a core component of the Microsoft Windows operating system and enable the creation and management of long-running processes.
* The services manage a wide variety of functions including network connections, speaker sound, data backup, user credentials and display colors. Windows Services perform a similar function as UNIX daemons.
* Windows Services are managed via the Services Control Manager panel. The panel shows a list of services and for each, name, description, status (running, stopped or paused) and the type of service.
* You can stop, pause, start, delay start, or resume each service as appropriate. You can also modify the start mechanism (Manual or Automatic) or specify an account.
* Windows Services broadly fall into three categories depending on the actions and applications they control:
  + Local Services
  + Network Services
  + System services
* Services can be deleted by a user with administrative privileges.
* Windows Services do not have a user interface; they run in the background and the user does not directly interact with them. A Windows Service does not stop when a user logs off the computer.
* Windows Services usually run under administrative privileges even when a non-administrator user is logged in and using the computer. The average Windows Service has more control over the machine compared to a regular application.

Examples of Windows Services:

* + **Active Directory Service** – Active Directory is a service Microsoft developed for Windows networks. It is included by default in most Microsoft Windows Server systems. Active Directory oversees centralized domain management and identity-related functions.
  + **DNS Client Service** – This service resolves domain names to IP addresses and locally caches this data.
  + **Computer Browser Service** – It allows users to easily locate shared resources on neighboring computers. All information is aggregated on one of the computers (referred to as the Master Browser) and other computers contact this machine for information on shared resources.
  + **Internet Connection Sharing (ICS) Service –** ICS enables the use of one device connected to the internet as an access point for other devices. Access could be through Ethernet broadband, cellular service or other gateway.
  + **Routing and Remote Access Service** – This service makes it possible to create applications that manage the remote access and routing capabilities of the Windows operating system. It allows the machine to act as a network router.

**6.Creating Process:**

The **CreateProcess** function creates a new process and its primary thread. The new process executes the specified executable file.

**BOOL CreateProcess(**

**LPCTSTR** *lpApplicationName***,**

// pointer to name of executable module

**LPTSTR** *lpCommandLine***,** // pointer to command line string

**LPSECURITY\_ATTRIBUTES** *lpProcessAttributes***,** // process security attributes

**LPSECURITY\_ATTRIBUTES** *lpThreadAttributes***,** // thread security attributes

**BOOL** *bInheritHandles***,** // handle inheritance flag

**DWORD** *dwCreationFlags***,** // creation flags

**LPVOID** *lpEnvironment***,** // pointer to new environment block

**LPCTSTR** *lpCurrentDirectory***,** // pointer to current directory name

**LPSTARTUPINFO** *lpStartupInfo***,** // pointer to STARTUPINFO

**LPPROCESS\_INFORMATION** *lpProcessInformation* // pointer to PROCESS\_INFORMATION

**);**

**Proc.cpp**

#include <windows.h>

#include <stdio.h>

#include <tchar.h>

int \_tmain( int argc, TCHAR \*argv[] )

{

    STARTUPINFO si;

    PROCESS\_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );

    si.cb = sizeof(si);

    ZeroMemory( &pi, sizeof(pi) );

    if( argc != 2 )

    {

        printf("Usage: %s [cmdline]\n", argv[0]);

        return 0;

    }

    // Start the child process.

    if( !CreateProcess( NULL,   // No module name (use command line)

        argv[1],        // Command line

        NULL,           // Process handle not inheritable

        NULL,           // Thread handle not inheritable

        FALSE,          // Set handle inheritance to FALSE

        0,              // No creation flags

        NULL,           // Use parent's environment block

        NULL,           // Use parent's starting directory

        &si,            // Pointer to STARTUPINFO structure

        &pi )           // Pointer to PROCESS\_INFORMATION structure

    )

    {

        printf( "CreateProcess failed (%d).\n", GetLastError() );

        return 0;

    }

    // Wait until child process exits.

    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles.

    CloseHandle( pi.hProcess );

    CloseHandle( pi.hThread );

    return 0;

}

**Hello.cpp**

#include <bits/stdc++.h>

using namespace std;

int main(){

    ofstream file;

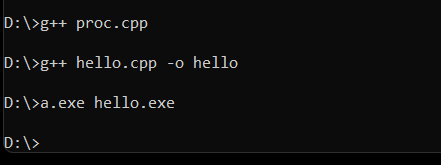
    file.open("data.txt",ios::out);

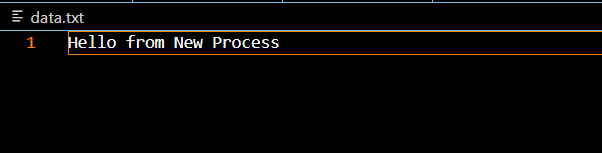
    file<<"Hello from New Process";

    file.close();

    return 0;

}

****

****

**7.Difference b/w Process and Service**

|  |  |
| --- | --- |
| **Process** | **Service** |
| * A process is an instance of a computer program or an app * It can be executed using one or more threads. One program can run multiple processes. * processes are directly related to the app we are currently running on our Windows computer, they have a user interface to interact with, and are usually manually started. * process exits when you close the associated app. * a process is a user service (designed for the end user) | * computer program that work in the background. * they are not exactly made, or running, for the end users, but helps the programs perform their tasks. * Services do not have a user interface. * A service is usually started by the operating system itself. However, a user can start or restart it manually. * Service continues to run in the background even though it has an associated process with a user interface that is shut. * a service is a Windows service (designed for Windows OS). |