

```
// InterfaceEnum.cpp: Registered interface enumeration sample
// Copyright (C) 2000 by Walter Oney
// All rights reserved
```

```
#include "stdafx.h"
```

```
BOOL GuidFromString(GUID* guid, LPCTSTR string);
BOOL htol(LPCTSTR& string, PDWORD presult);
BOOL htos(LPCTSTR& string, PWORD presult);
BOOL htob(LPCTSTR& string, PBYTE presult);
LPCTSTR InterfaceGuidName(GUID* guid);
```

```
#define arraysize(p) (sizeof(p)/sizeof((p)[0]))
```

```
////////////////////////////////////
```

```
int main(int argc, char* argv[])
```

```
{ // main
```

```
// Registered device interfaces have persistent registry keys below
// HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\DeviceClasses.
// Enumerate those keys to learn which device interfaces have been
// registered by any device.
```

```
HKEY hkey;
```

```
DWORD code = RegOpenKey(HKEY_LOCAL_MACHINE, REGSTR_PATH_DEVICE_CLASSES, &hkey);
```

```
if (code)
```

```
return code;
```

```
TCHAR keyname[256];
```

```
for (DWORD keyindex = 0; RegEnumKey(hkey, keyindex, keyname, 256) == 0; ++keyindex)
```

```
{ // for each registered interface
```

```
// Convert the key name to a GUID. Seems like there ought to be an SDK
// function that would do this without needing a UNICODE string first...
```

```
GUID interfaceguid;
```

```

if (!GuidFromString(&interfaceguid, keyname))
    continue;          // can't convert key to GUID

// Print the header for a section of interface reports

printf(_T("\n%s (%s):\n"), keyname, InterfaceGuidName(&interfaceguid));

// Open a device information set to enumerate instances of this interface

HDEVINFO infoset = SetupDiGetClassDevs(&interfaceguid, NULL, NULL, DIGCF_DEVICEINTERFACE | DIGCF_PRESENT);
if (infoset == INVALID_HANDLE_VALUE)
    {
        // no devices
        _tprintf(_T(" (No devices)\n"));
        continue;
    }
    // no devices

// Report information about these devices

SP_DEVICE_INTERFACE_DATA interfacedata = {sizeof(SP_DEVICE_INTERFACE_DATA)};
for (DWORD devindex = 0; SetupDiEnumDeviceInterfaces(infoset, NULL, &interfaceguid, devindex, &interfacedata); ++devindex)
    {
        // for each device

        // Obtain information about the device. The following function call is expected
        // to fail because we're not providing a buffer for the so-called "detail"
        // information. The "detail" is just the symbolic link name, which we're not
        // interested in here.

        SP_DEVINFO_DATA devicedata = {sizeof(SP_DEVINFO_DATA)};
        if (SetupDiGetDeviceInterfaceDetail(infoset, &interfacedata, NULL, 0, NULL, &devicedata)
            || GetLastError() != ERROR_INSUFFICIENT_BUFFER)
            continue;          // unexpected result

        // Determine the friendly name parameter (if any) for the interface instance. This is useful
        // when displaying kernel-streaming interface information

        BYTE interfacename[512];

```

```

interfacename[0] = 0;
HKEY interfacekey = SetupDiOpenDeviceInterfaceRegKey(InfoSet, &interfacedata, 0, KEY_READ);
if (interfacekey)
    {
        // look for friendly name
        DWORD size = sizeof(interfacename);
        RegQueryValueEx(interfacekey, _T("FriendlyName"), 0, NULL, interfacename, &size);
        RegCloseKey(interfacekey);
        if (interfacename[0])
            _tcsat((LPTSTR) interfacename, _T("/"));
    }
    // look for friendly name

// Determine the friendly name or description of this device

BYTE devname[512];
if (SetupDiGetDeviceRegistryProperty(InfoSet, &devicedata, SPDRP_FRIENDLYNAME, NULL, devname, sizeof(devname), NULL)
    || SetupDiGetDeviceRegistryProperty(InfoSet, &devicedata, SPDRP_DEVICEDESC, NULL, devname, sizeof(devname), NULL
))
    _tprintf(_T("    %s%s\n"), interfacename, devname);
}
// for each device

SetupDiDestroyDeviceInfoList(InfoSet);

if (devindex == 0)
    {
        // no devices
        _tprintf(_T(" (No devices)\n"));
        continue;
    }
    // no devices
// for each registered interface

RegCloseKey(hkey);

return 0;
}
// main

////////////////////////////////////
// GuidFromString converts a string of the form {c200e360-38c5-11ce-ae62-08002b2b79ef}
// into a GUID structure. It return TRUE if the conversion was successful

```

```

BOOL GuidFromString(GUID* guid, LPCTSTR string)
{
    // GuidFromString
    return *string++ == _T('{')
        && htol(string, &guid->Data1)
        && *string++ == _T('-')
        && htos(string, &guid->Data2)
        && *string++ == _T('-')
        && htos(string, &guid->Data3)
        && *string++ == _T('-')
        && htob(string, &guid->Data4[0])
        && htob(string, &guid->Data4[1])
        && *string++ == _T('-')
        && htob(string, &guid->Data4[2])
        && htob(string, &guid->Data4[3])
        && htob(string, &guid->Data4[4])
        && htob(string, &guid->Data4[5])
        && htob(string, &guid->Data4[6])
        && htob(string, &guid->Data4[7])
        && *string++ == _T('}')
        && *string == 0;
}
// GuidFromString

////////////////////////////////////
// Hexadecimal conversion routines for use by GuidFromString

BOOL htol(LPCTSTR& string, PDWORD result)
{
    // htol
    DWORD result = 0;
    for (int i = 0; i < 8; ++i)
    {
        // convert hex digits
        TCHAR ch = *string++;
        BYTE hexit;
        if (ch >= _T('0') && ch <= _T('9'))
            hexit = ch - _T('0');
        else if (ch >= _T('A') && ch <= _T('F'))
            hexit = ch - (_T('A') - 10);
    }
}

```

```

    else if (ch >= _T('a') && ch <= _T('f'))
        hexit = ch - (_T('a') - 10);
    else
        return FALSE;

    result <<= 4;
    result |= hexit;
} // convert hex digits

*presult = result;
return TRUE;
} // htol

BOOL htos(LPCTSTR& string, PWORD presult)
{ // htos
WORD result = 0;
for (int i = 0; i < 4; ++i)
{ // convert hex digits
TCHAR ch = *string++;
BYTE hexit;
if (ch >= _T('0') && ch <= _T('9'))
    hexit = ch - _T('0');
else if (ch >= _T('A') && ch <= _T('F'))
    hexit = ch - (_T('A') - 10);
else if (ch >= _T('a') && ch <= _T('f'))
    hexit = ch - (_T('a') - 10);
else
    return FALSE;

    result <<= 4;
    result |= hexit;
} // convert hex digits

*presult = result;
return TRUE;
} // htos

```

```

BOOL htob(LPCTSTR& string, PBYTE result)
{
    // htob
    BYTE result = 0;
    for (int i = 0; i < 2; ++i)
    {
        // convert hex digits
        TCHAR ch = *string++;
        BYTE hexit;
        if (ch >= _T('0') && ch <= _T('9'))
            hexit = ch - _T('0');
        else if (ch >= _T('A') && ch <= _T('F'))
            hexit = ch - (_T('A') - 10);
        else if (ch >= _T('a') && ch <= _T('f'))
            hexit = ch - (_T('a') - 10);
        else
            return FALSE;

        result <<= 4;
        result |= hexit;
    }
    // convert hex digits

    *result = result;
    return TRUE;
}
// htob

////////////////////////////////////
// InterfaceGuidName returns a symbolic name for an interface GUID

LPCTSTR InterfaceGuidName(GUID* guid)
{
    // InterfaceGuidName

    // Define a table of standard interface guids. This first redefinition of
    // DEFINE_GUID is the standard one followed by a semicolon.

    #undef DEFINE_GUID
    #define DEFINE_GUID(name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8) \
        GUID name = { l, w1, w2, { b1, b2, b3, b4, b5, b6, b7, b8 } } ;

```

```

#include "StandardInterfaces.h"

// Now define a lookup table for them. This second redefinition of
// DEFINE_GUID generates the table

#undef DEFINE_GUID
#define DEFINE_GUID(name, l, w1, w2, b1, b2, b3, b4, b5, b6, b7, b8) {&name, TEXT(#name)},

struct {GUID* guid; LPCTSTR name;} guidname[] = {
    #include "StandardInterfaces.h"
};

for (int i = 0; i < arraysize(guidname); ++i)
    if (*guidname[i].guid == *guid)
        return guidname[i].name;

return _T("Unknown Interface GUID");
} // InterfaceGuidName

```