Босько Виктор, 151004, Лабораторная работа №4

**Source.cpp(lab4)**

#include <Windows.h>

#include <iostream>

#include "Key.h"

int main()

{

Key key(HKEY\_CURRENT\_USER, L"lab4\\Key");

const BYTE data[]{ 100, 82, 16, 5, 19, 55, 15, 101 };

int dataSize = 8;

key.AddValue(L"123", REG\_QWORD, data, dataSize);

key.Save();

key.ReadFlags(L"123");

RegCloseKey(key.GetKey());

key.Find(HKEY\_CURRENT\_USER, L"lab4\\Key");

return 0;

}

**Key.h(lab4)**

#pragma once

#include <Windows.h>

#include <iostream>

class Key

{

public:

// Initializes key by head directory name and subdirectory name

Key(HKEY hKey, LPCWSTR lpSubKey);

// Reads value of flag

bool ReadFlags(LPCWSTR lpValueName);

// Adds another value if it not exists or changes it

bool AddValue(LPCWSTR valueName, DWORD type, const BYTE\* data, DWORD size);

// Saves key changes

bool Save();

// Returs key handler

HKEY GetKey();

// Open any key by name

static HKEY Open(HKEY hKey, LPCWSTR lpSubKey);

// Find key by name

static bool Find(HKEY hKey, LPCWSTR lpSubKey);

private:

// Key value

HKEY key;

// Subkey name

LPCWSTR subKey;

// Prints type of key

void PrintType(DWORD type);

};

**Key.cpp(lab4)**

#include "Key.h"

Key::Key(HKEY hKey, LPCWSTR lpSubKey)

{

DWORD keyDisposition;

this->key = hKey;

subKey = lpSubKey;

LSTATUS status = RegCreateKeyEx(hKey, lpSubKey, NULL, NULL, NULL,

KEY\_READ | KEY\_WRITE, NULL, &key, &keyDisposition);

if (status == ERROR\_SUCCESS)

{

if (keyDisposition == REG\_OPENED\_EXISTING\_KEY)

{

std::cout << "Opened existing key. No changes" << std::endl;

}

if (keyDisposition == REG\_CREATED\_NEW\_KEY)

{

std::cout << "Created new key" << std::endl;

}

}

else

{

key = NULL;

std::cout << "Error. Key was not created" << std::endl;

}

}

HKEY Key::GetKey()

{

return key;

}

HKEY Key::Open(HKEY hKey, LPCWSTR lpSubKey)

{

HKEY res;

if (RegOpenKeyEx(hKey, lpSubKey, NULL, KEY\_READ, &res) == ERROR\_SUCCESS)

{

return res;

}

return NULL;

}

bool Key::Find(HKEY hKey, LPCWSTR lpSubKey)

{

HKEY res;

if (RegOpenKeyEx(hKey, lpSubKey, NULL, KEY\_READ, &res) == ERROR\_SUCCESS)

{

RegCloseKey(res);

std::cout << "Found it: " << (int)res << std::endl;

return true;

}

return false;

}

bool Key::ReadFlags(LPCWSTR lpValueName)

{

BYTE buffer[256];

// Must contain initial buffer size

DWORD bufLength = 256;

DWORD type = 0;

// Changes buffer size to real, initializaes buffer and type

if (RegQueryValueEx(key, lpValueName, NULL, &type, buffer, &bufLength) == ERROR\_SUCCESS)

{

PrintType(type);

for (int i = 0; i < bufLength; i++)

{

std::cout << (int)buffer[i] << " ";

}

std::cout << std::endl;

}

else

{

return false;

}

return true;

}

bool Key::AddValue(LPCWSTR valueName, DWORD type, const BYTE\* data, DWORD size)

{

return RegSetValueEx(key, valueName, NULL, type, data, size) == ERROR\_SUCCESS;

}

bool Key::Save()

{

if (RegFlushKey(key) == ERROR\_SUCCESS) {

std::cout << "Key saved" << std::endl;

return true;

}

else {

std::cout << "Saving error" << std::endl;

return false;

}

}

void Key::PrintType(DWORD type)

{

int typesSize = 11;

DWORD types[]

{

REG\_BINARY, REG\_DWORD, REG\_DWORD\_LITTLE\_ENDIAN, REG\_DWORD\_BIG\_ENDIAN,

REG\_EXPAND\_SZ, REG\_LINK, REG\_MULTI\_SZ, REG\_NONE, REG\_QWORD, REG\_QWORD\_LITTLE\_ENDIAN, REG\_SZ

};

std::string messages[]

{

"Binary data in any form",

"A 32-bit number",

"A 32-bit number in little-endian format", // == DWORD

"A 32-bit number in big-endian format",

"A null-terminated string that contains unexpanded references to environment variables",

"A null-terminated Unicode string that contains the target path of a symbolic link that was created by calling the RegCreateKeyEx function with REG\_OPTION\_CREATE\_LINK",

"A sequence of null-terminated strings, terminated by an empty string",

"No defined value type",

"A 64-bit number",

"A 64-bit number in little-endian format", // == QWORD

"A null-terminated string. It's either a Unicode or an ANSI string, depending on whether you use the Unicode or ANSI functions"

};

for (int i = 0; i < typesSize; i++)

{

if ((type ^ types[i]) == 0)

{

std::cout << messages[i] << std::endl;

}

}

}

**Source.cpp(lab4\_additional)**

#include <Windows.h>

#include <iostream>

#define REG\_NOTIFY\_CHAGES REG\_NOTIFY\_CHANGE\_NAME | REG\_NOTIFY\_CHANGE\_ATTRIBUTES | REG\_NOTIFY\_CHANGE\_LAST\_SET | REG\_NOTIFY\_CHANGE\_SECURITY

// If registy key changed, notifies user about it

bool MonitorRegistryChanges(HKEY hKey)

{

HKEY hMonitorKey;

bool isMonitoring = true;

if (RegOpenKeyEx(hKey, NULL, 0, KEY\_NOTIFY, &hMonitorKey) == ERROR\_SUCCESS)

{

std::cout << "Register change detection started..." << std::endl;

while (isMonitoring)

{

if (RegNotifyChangeKeyValue(hMonitorKey, TRUE, REG\_NOTIFY\_CHAGES, NULL, FALSE) == ERROR\_SUCCESS)

{

std::cout << "Key's change detected" << std::endl;

}

else

{

std::cout << "Error during key monitoring" << std::endl;

isMonitoring = false;

}

}

RegCloseKey(hMonitorKey);

return true;

}

else

{

std::cout << "Open register key error" << std::endl;

return false;

}

}

int main()

{

HKEY hKey;

if (RegOpenKeyEx(HKEY\_CURRENT\_USER, L"lab4\\Key", 0, KEY\_SET\_VALUE, &hKey) == ERROR\_SUCCESS)

{

MonitorRegistryChanges(hKey);

RegCloseKey(hKey);

}

else

{

std::cout << "Open register key error" << std::endl;

}

return 0;

}