Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №5

на тему

Реестр и журналы (Windows). Доступ к реестру Windows. Работа с журналами Windows. Другие вспомогательные средства управления

Студент: гр.153502

Макаренко А.И.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495008)

[2 Теоретические сведения 4](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495009)

[3 Описание функций программы 6](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495010)

[Список использованных источников 8](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495011)

[Приложение А 9](file:///C:\Users\alyona\Downloads\Telegram%20Desktop\ОСиСП%201%20ЛР.docx#_Toc147495012)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание приложения, использующее реестр *Windows*.

В качестве задачи необходимо создать утилиту для массовой установки и удаления программ с использованием информации из реестра Windows.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Реестр – это определяемая системой база данных, в которой приложения и системные компоненты хранят и извлекают данные конфигурации. Ключ может иметь любое количество значений, а значения могут быть в любой форме. У каждого ключа есть имя, состоящее из одного или нескольких печатных символов. В именах ключей регистр не учитывается. Имена ключей не могут содержать символ обратной косой черты (\), но можно использовать любой другой печатный символ. Имена значений и данные могут содержать символ обратной косой черты [1].

*Hive* – это логическая группа ключей, подразделов и значений в реестре, которая содержит набор вспомогательных файлов, загруженных в память при запуске операционной системы или при входе пользователя в систему [2].

Все ключи, которые считаются кустами, начинаются с «*HKEY*» и находятся в корне или на вершине иерархии реестра. Разница между кустом реестра и ключом реестра заключается в том, что куст реестра — это первая папка в реестре, содержащая ключи реестра, тогда как ключи реестра — это папки внутри кустов, содержащие значения реестра и другие ключи реестра [3].

Значение реестра может хранить данные в одном из нескольких форматов, например строковое или целочисленное значение. Основные из них это *REG\_BINARY, REG\_DWORD, REG\_QWORD, REG\_SZ*.

Для работы с ключами и значениями реестра использовались функции, описанные в заголовочном файле *winreg.h*. Для реализации использовались функции *RegOpenKeyEx, RegEnumKeyEx, RegQueryValueEx*.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Для реализации задачи было выбрано приложение, реализующее массовую установку и удаление программ с использованием реестра *Windows*. Согласно формулировке задачи, были спроектированы следующие функции программы:

– установка приложений;

– удаление приложений;

– использование реестра для поиска пути удаления.

При запуске программа отображает две области. Область слева отображает установленные приложения, область справа – доступные приложения для установки. Интерфейс приложения показан на рисунке 3.1.

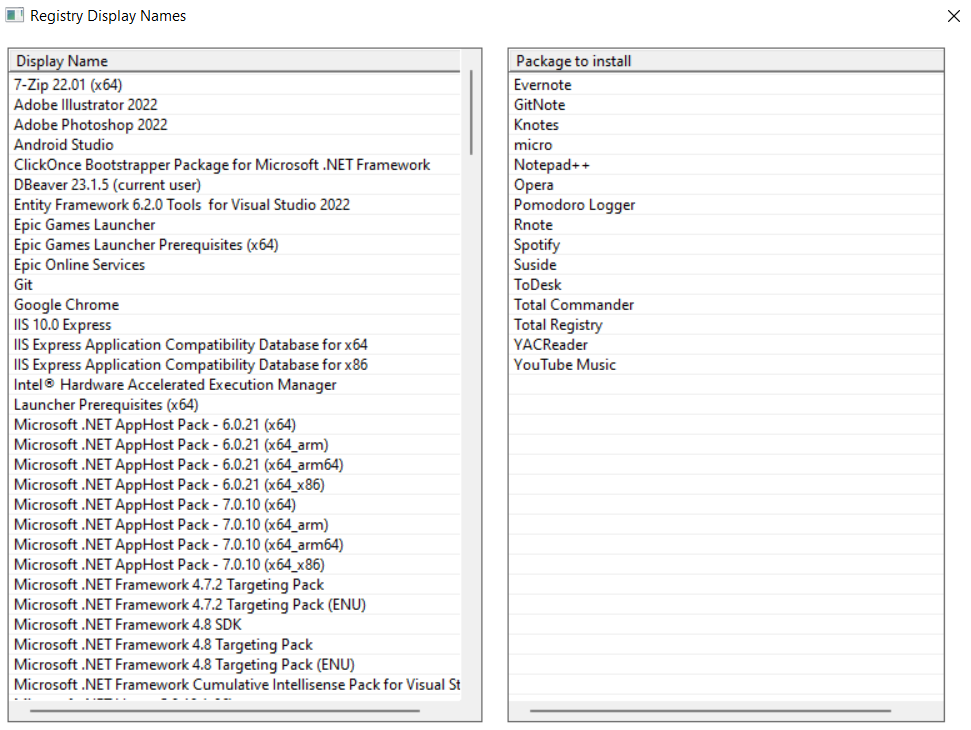


Рисунок 3.1 – Окно приложения

По клику правой кнопкой мыши отображается контекстное меню. Вид контекстного меню в области установленных приложений показан на рисунке 3.2, а вид контекстного меню в области приложений, доступных для установки, отображён на рисунке 3.3.

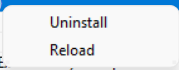


Рисунок 3.2 – Вид контекстного меню в области установленных приложений



Рисунок 3.3 – Вид контекстного меню в области доступных приложений

Для установки приложения из списка доступных необходимо нажать в контекстном меню соответствующей области кнопку *Install*. После этого откроется окно для подтверждения установки. Данное окно представлено на рисунке 3.4. Чтобы продолжить и начать установку необходимо нажать кнопку *OK*.

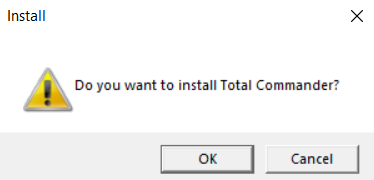


Рисунок 3.4 – Окно подтверждения установки

После подтверждения откроется консоль, отображающая процесс установки. Консоль представлена на рисунке 3.5. Во время установки одного приложения можно параллельно запустить установку ещё одного.

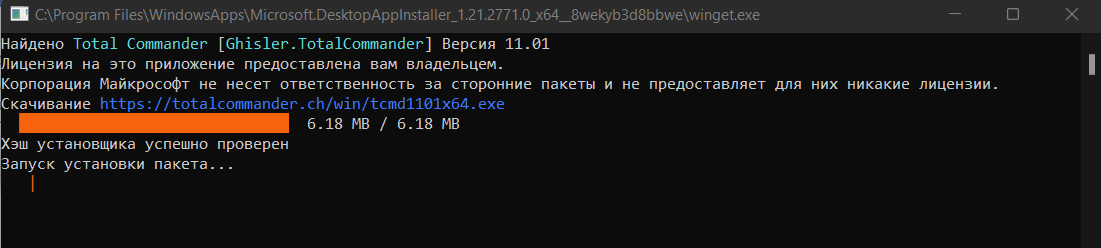


Рисунок 3.5 – Консоль с процессом установки

Для того, чтобы установленное приложение отобразилось в списке установленных, необходимо в контекстном меню соответствующей области нажать кнопку *Reload*. После этого приложение отобразится в списке установленных.

Чтобы удалить приложение необходимо в контекстном меню области установленных приложений нажать кнопку *Uninstall*. Появится окно с подтверждением удаления. Данное окно представлено на рисунке 3.6. Чтобы продолжить и удалить приложение необходимо нажать кнопку *OK*.

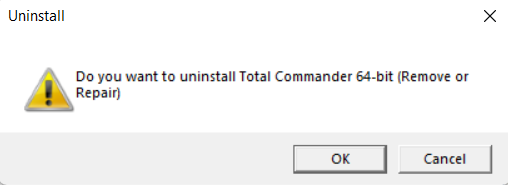


Рисунок 3.6 – Окно подтверждения удаления

После подтверждения запустится программа удаления данного приложения. Пример такой программы представлен на рисунке 3.7.

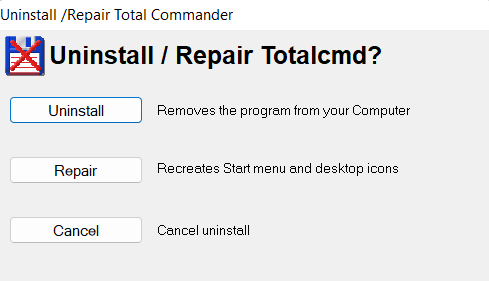


Рисунок 3.7 – Программа удаления приложения *Total Commander*

После удаления появится окно, в котором говорится, что приложение было успешно удалено. Данное окно представлено на рисунке 3.8.

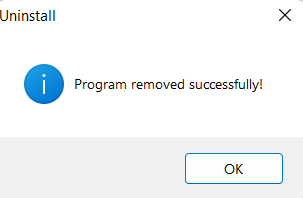


Рисунок 3.8 – Окно завершения удаления

Чтобы удалённое приложение пропало из списка установленных, нужно нажать в контекстном меню для установленных приложений кнопку *Reload*.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Реестр [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry>.

[2] Кусты реестра [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sysinfo/registry-hives>.

[3] Что такое кусты реестра [Электронный ресурс]. – Режим доступа: <https://www.lifewire.com/what-is-a-registry-hive-2625986>.

ПРИЛОЖЕНИЕ А

**Исходный код программы**

**Файл Lab5.cpp**

#include <Windows.h>

#include <CommCtrl.h>

#include <string>

#include <algorithm>

#pragma comment(lib, "comctl32.lib")

#include "uninstall.h"

#include "install.h"

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {

switch (message) {

case WM\_CREATE:

{

CreateUninstallListView(hWnd, lParam);

CreateInstallListView(hWnd, lParam);

return DefWindowProc(hWnd, message, wParam, lParam);

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDM\_UNINSTALL:

HandleUninstall(hWnd);

break;

case IDM\_UNINSTALL\_RELOAD:

PopulateWholeUninstallListView(g\_hUninstallListView);

break;

case IDM\_INSTALL:

HandleInstall(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

break;

case WM\_NOTIFY:

if (((LPNMHDR)lParam)->idFrom == ID\_UNINSTALL\_LISTVIEW && ((LPNMHDR)lParam)->code == NM\_RCLICK) {

POINT cursorPos;

GetCursorPos(&cursorPos);

ShowUninstallContextMenu(hWnd, cursorPos.x, cursorPos.y);

}

if (((LPNMHDR)lParam)->idFrom == ID\_INSTALL\_LISTVIEW && ((LPNMHDR)lParam)->code == NM\_RCLICK) {

POINT cursorPos;

GetCursorPos(&cursorPos);

ShowInstallContextMenu(hWnd, cursorPos.x, cursorPos.y);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow) {

INITCOMMONCONTROLSEX icex;

icex.dwSize = sizeof(INITCOMMONCONTROLSEX);

icex.dwICC = ICC\_LISTVIEW\_CLASSES;

InitCommonControlsEx(&icex);

WNDCLASSW wc = { 0 };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.lpszClassName = L"RegistryDisplayNames";

RegisterClassW(&wc);

HWND hWnd = CreateWindowW(L"RegistryDisplayNames", L"Registry Display Names",

WS\_SYSMENU | WS\_CAPTION, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, NULL, NULL, hInstance, NULL);

if (hWnd == NULL) {

MessageBoxW(NULL, L"Window creation failed.", L"Error", MB\_ICONERROR);

return 1;

}

ShowWindow(hWnd, nCmdShow);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

**Файл install.h**

#pragma once

#include <Windows.h>

#include <CommCtrl.h>

#include <string>

#include <vector>

#pragma comment(lib, "comctl32.lib")

#define ID\_INSTALL\_LISTVIEW 546

#define IDM\_INSTALL 547

HWND g\_hInstallListView;

struct InstallData {

std::wstring displayName;

std::wstring packageName;

InstallData(const std::wstring& displayName, const std::wstring& packageName) : displayName(displayName), packageName(packageName) {}

};

void WingetInstall(std::wstring packageName) {

std::wstring command = L"install -e --id " + packageName;

SHELLEXECUTEINFO ShExecInfo = { 0 };

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.lpVerb = L"open";

ShExecInfo.nShow = SW\_SHOWNORMAL;

ShExecInfo.lpFile = L"winget";

ShExecInfo.lpParameters = command.c\_str();

ShellExecuteEx(&ShExecInfo);

}

void HandleInstall(HWND hMainWindow) {

int selectedItem = ListView\_GetNextItem(g\_hInstallListView, -1, LVNI\_SELECTED);

if (selectedItem != -1) {

LVITEMW lvItem = { 0 };

lvItem.mask = LVIF\_PARAM;

lvItem.iItem = selectedItem;

lvItem.iSubItem = 0;

ListView\_GetItem(g\_hInstallListView, &lvItem);

InstallData\* lparam = (InstallData\*)lvItem.lParam;

std::wstring message = L"Do you want to install " + lparam->displayName + L"?";

DWORD res = MessageBoxW(hMainWindow, message.c\_str(), L"Install", MB\_ICONWARNING | MB\_OKCANCEL);

if (res == IDOK) {

WingetInstall(lparam->packageName);

}

}

}

void PopulatInstallListView(HWND hListView) {

std::vector<InstallData> items{

InstallData(L"Total Registry", L"PavelYosifovich.TotalRegistry"),

InstallData(L"Pomodoro Logger", L"zxch3n.PomodoroLogger"),

InstallData(L"Total Commander", L"Ghisler.TotalCommander"),

InstallData(L"micro", L"zyedidia.micro"),

InstallData(L"Suside", L"zeankundev.suside"),

InstallData(L"GitNote", L"zhaopengme.gitnote"),

InstallData(L"ToDesk", L"Youqu.ToDesk"),

InstallData(L"YouTube Music", L"Ytmdesktop.Ytmdesktop"),

InstallData(L"YACReader", L"YACReader.YACReader"),

InstallData(L"Spotify", L"Spotify.Spotify"),

InstallData(L"Opera", L"Opera.OperaGX"),

InstallData(L"Evernote", L"evernote.evernote"),

InstallData(L"Rnote", L"flxzt.rnote"),

InstallData(L"Notepad++", L"Notepad++.Notepad++"),

InstallData(L"Knotes", L"1MHz.Knotes"),

};

ListView\_DeleteAllItems(hListView);

for (auto& data : items) {

LVITEMW lvItem = { 0 };

lvItem.mask = LVIF\_TEXT | LVIF\_PARAM;

lvItem.iItem = 0;

lvItem.iSubItem = 0;

lvItem.pszText = (LPWSTR)data.displayName.c\_str();

InstallData\* lparam = new InstallData(data);

lvItem.lParam = (LPARAM)lparam;

ListView\_InsertItem(hListView, &lvItem);

}

}

HMENU CreateInstallContextMenu() {

HMENU hMenu = CreatePopupMenu();

AppendMenuW(hMenu, MF\_STRING, IDM\_INSTALL, L"Install");

return hMenu;

}

void ShowInstallContextMenu(HWND hWnd, int xPos, int yPos) {

HMENU hContextMenu = CreateInstallContextMenu();

TrackPopupMenu(hContextMenu, TPM\_TOPALIGN | TPM\_LEFTALIGN, xPos, yPos, 0, hWnd, NULL);

DestroyMenu(hContextMenu);

}

void CreateInstallListView(HWND hWnd, LPARAM lParam) {

g\_hInstallListView = CreateWindow(WC\_LISTVIEW, L"List View",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | LVS\_REPORT | LVS\_SORTASCENDING, 410, 10, 350, 540,

hWnd, (HMENU)ID\_INSTALL\_LISTVIEW, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

if (g\_hInstallListView == NULL) {

MessageBoxW(NULL, L"List view creation failed.", L"Error", MB\_ICONERROR);

return;

}

ListView\_SetExtendedListViewStyle(g\_hInstallListView, LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES | LVS\_EX\_DOUBLEBUFFER);

LVCOLUMNW lvCol;

lvCol.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvCol.cx = 380;

lvCol.pszText = (LPWSTR)L"Package to install";

ListView\_InsertColumn(g\_hInstallListView, 0, &lvCol);

PopulatInstallListView(g\_hInstallListView);

}

**Файл uninstall.h**

#pragma once

#include <Windows.h>

#include <CommCtrl.h>

#include <string>

#include <algorithm>

#pragma comment(lib, "comctl32.lib")

#define IDM\_UNINSTALL 13

#define IDM\_UNINSTALL\_RELOAD 14

#define ID\_UNINSTALL\_LISTVIEW 153

struct UninstallData {

std::wstring displayName;

std::wstring uninstallString;

};

HWND g\_hUninstallListView;

void PopulateUninstallListView(HWND hListView, HKEY hive, std::wstring registryPath) {

HKEY hKey;

if (RegOpenKeyExW(hive, registryPath.c\_str(), 0, KEY\_READ, &hKey) == ERROR\_SUCCESS) {

DWORD index = 0;

WCHAR subkeyName[255];

DWORD subkeyNameSize = sizeof(subkeyName);

while (RegEnumKeyExW(hKey, index, subkeyName, &subkeyNameSize, NULL, NULL, NULL, NULL) == ERROR\_SUCCESS) {

HKEY subKey;

if (RegOpenKeyExW(hKey, subkeyName, 0, KEY\_READ, &subKey) == ERROR\_SUCCESS) {

std::wstring displayName;

displayName.resize(1024);

std::wstring uninstallString;

uninstallString.resize(1024);

DWORD dislpayNameStringSize = 1024;

DWORD uninstallStringSize = 1024;

if (RegQueryValueExW(subKey, L"DisplayName", NULL, NULL, (LPBYTE)&displayName[0], &dislpayNameStringSize) == ERROR\_SUCCESS

&& RegQueryValueExW(subKey, L"UninstallString", NULL, NULL, (LPBYTE)&uninstallString[0], &uninstallStringSize) == ERROR\_SUCCESS) {

std::wstring copy = uninstallString;

std::transform(copy.begin(), copy.end(), copy.begin(), tolower);

if (copy.rfind(L"msiexec.exe /i") != 0) {

LVITEMW lvItem = { 0 };

lvItem.mask = LVIF\_TEXT | LVIF\_PARAM;

lvItem.iItem = index;

lvItem.iSubItem = 0;

lvItem.pszText = (LPWSTR)displayName.c\_str();

UninstallData\* lparam = new UninstallData();

lparam->uninstallString = uninstallString;

lparam->displayName = displayName;

lvItem.lParam = (LPARAM)lparam;

ListView\_InsertItem(hListView, &lvItem);

}

}

RegCloseKey(subKey);

}

index++;

subkeyNameSize = sizeof(subkeyName);

}

RegCloseKey(hKey);

}

}

HMENU CreateUninstallContextMenu() {

HMENU hMenu = CreatePopupMenu();

AppendMenuW(hMenu, MF\_STRING, IDM\_UNINSTALL, L"Uninstall");

AppendMenuW(hMenu, MF\_STRING, IDM\_UNINSTALL\_RELOAD, L"Reload");

return hMenu;

}

void ShowUninstallContextMenu(HWND hWnd, int xPos, int yPos) {

HMENU hContextMenu = CreateUninstallContextMenu();

TrackPopupMenu(hContextMenu, TPM\_TOPALIGN | TPM\_LEFTALIGN, xPos, yPos, 0, hWnd, NULL);

DestroyMenu(hContextMenu);

}

void ExecuteUninstallCommand(const std::wstring& filepath) {

if (filepath.rfind(L"winget") == 0) {

int index = 7; // after winget

std::wstring command = filepath.substr(index, filepath.length() - index);

SHELLEXECUTEINFO ShExecInfo = { 0 };

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.lpVerb = L"open";

ShExecInfo.nShow = SW\_SHOWNORMAL;

ShExecInfo.lpFile = L"winget";

ShExecInfo.lpParameters = command.c\_str();

ShellExecuteEx(&ShExecInfo);

return;

}

int index = filepath.find(L".exe");

while (index < filepath.length() && filepath[index] != L' ') {

index++;

}

SHELLEXECUTEINFO ShExecInfo = { 0 };

ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);

ShExecInfo.fMask = SEE\_MASK\_NOCLOSEPROCESS;

ShExecInfo.lpVerb = L"open";

ShExecInfo.nShow = SW\_SHOWNORMAL;

if (index != filepath.length()) {

std::wstring fileLocation = filepath.substr(0, index);

std::wstring command = filepath.substr(index + 1, filepath.length() - (index + 1));

ShExecInfo.lpFile = fileLocation.c\_str();

ShExecInfo.lpParameters = command.c\_str();

ShellExecuteEx(&ShExecInfo);

}

else {

ShExecInfo.lpFile = filepath.c\_str();

ShellExecuteEx(&ShExecInfo);

}

}

void HandleUninstall(HWND hMainWindow) {

int selectedItem = ListView\_GetNextItem(g\_hUninstallListView, -1, LVNI\_SELECTED);

if (selectedItem != -1) {

LVITEMW lvItem = { 0 };

lvItem.mask = LVIF\_PARAM;

lvItem.iItem = selectedItem;

lvItem.iSubItem = 0;

ListView\_GetItem(g\_hUninstallListView, &lvItem);

UninstallData\* lparam = (UninstallData\*)lvItem.lParam;

std::wstring message = L"Do you want to uninstall " + lparam->displayName + L"?";

DWORD res = MessageBoxW(hMainWindow, message.c\_str(), L"Uninstall", MB\_ICONWARNING | MB\_OKCANCEL);

if (res == IDOK) {

ExecuteUninstallCommand(lparam->uninstallString);

}

}

}

void PopulateWholeUninstallListView(HWND listView) {

ListView\_DeleteAllItems(listView);

PopulateUninstallListView(g\_hUninstallListView, HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\WOW6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall");

PopulateUninstallListView(g\_hUninstallListView, HKEY\_LOCAL\_MACHINE, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall");

PopulateUninstallListView(g\_hUninstallListView, HKEY\_CURRENT\_USER, L"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Uninstall");

}

void CreateUninstallListView(HWND hWnd, LPARAM lParam) {

g\_hUninstallListView = CreateWindow(WC\_LISTVIEW, L"List View",

WS\_CHILD | WS\_VISIBLE | WS\_BORDER | LVS\_REPORT | LVS\_SORTASCENDING, 10, 10, 380, 540,

hWnd, (HMENU)ID\_UNINSTALL\_LISTVIEW, ((LPCREATESTRUCT)lParam)->hInstance, NULL);

if (g\_hUninstallListView == NULL) {

MessageBoxW(NULL, L"List view creation failed.", L"Error", MB\_ICONERROR);

return;

}

ListView\_SetExtendedListViewStyle(g\_hUninstallListView, LVS\_EX\_FULLROWSELECT | LVS\_EX\_GRIDLINES | LVS\_EX\_DOUBLEBUFFER);

LVCOLUMNW lvCol;

lvCol.mask = LVCF\_TEXT | LVCF\_WIDTH;

lvCol.cx = 380;

lvCol.pszText = (LPWSTR)L"Display Name";

ListView\_InsertColumn(g\_hUninstallListView, 0, &lvCol);

PopulateWholeUninstallListView(g\_hUninstallListView);

}