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 №2

на тему

Расширенное использование оконного интерфейса Win 32 и GDI.

Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook)

Студент: гр.153502 Сидорова У.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc145669228)

[2 Теоретические сведения 3](#_Toc145669228)

[3 Результат работы программы 4](#_Toc145669229)

[Список использованных источников 6](#_Toc145669230)

[Приложение A 7](#_Toc145669231)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание приложения на *Win32 API*, которое создает и использует элементы управления и реализует обработку различных сообщений.

В качестве задачи необходимо построить графический календарь с возможностью добавления и удаления событий.

**2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В лабораторной работе используются общие элементы управления, реализованные в общей библиотеке элементов управления, *Comctl32.dll*, которая является библиотекой *DLL*, входящей в состав операционной системы *Windows*. Как и другие окна элементов управления, общим элементом управления является дочернее окно, которое приложение использует вместе с другим окном для взаимодействия с пользователем [1].

Вывод результата происходит с помощью элемента *List View*. Элемент управления представлением списка — это окно, в котором отображается коллекция элементов. Элементы управления представлением списка предоставляют несколько способов упорядочивания и отображения элементов и являются гораздо более гибкими, чем простые списки [2].

Система передает входные данные в процедуру окна в виде сообщения. Сообщения создаются как системой, так и приложениями. Система создает сообщение при каждом входном событии. Система отправляет сообщение в оконную процедуру с набором из четырех параметров: дескриптор окна, идентификатор сообщения и два значения, называемые параметрами сообщения. Дескриптор окна определяет окно, для которого предназначено сообщение. Система использует его, чтобы определить, какая оконная процедура должна получить сообщение [3].

3 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

При запуске программы открывается оконное приложение (рисунок 1).

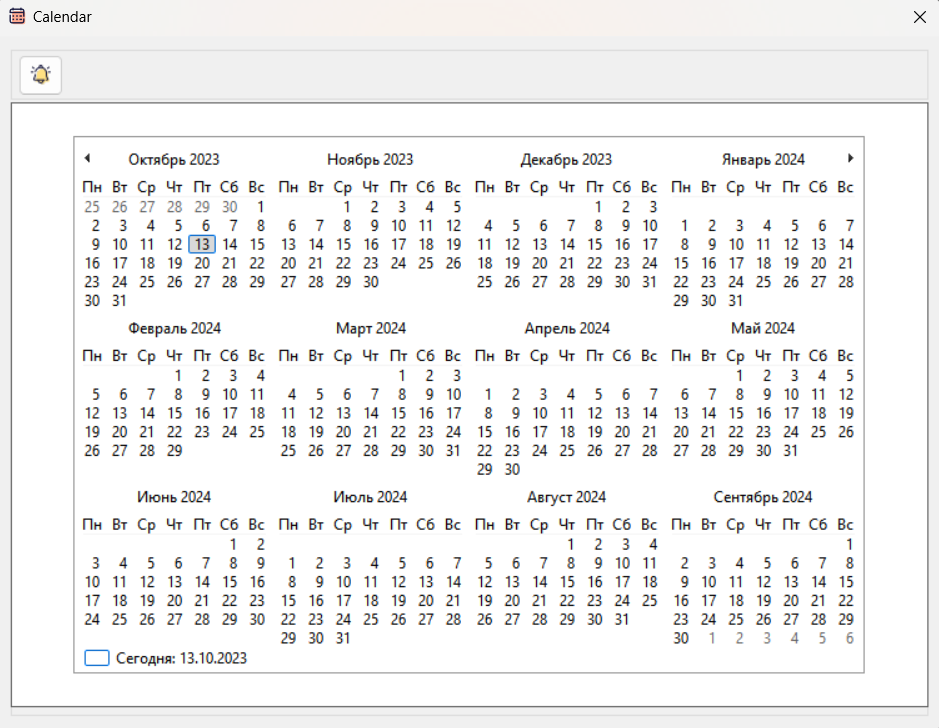


Рисунок 1 – Графический интерфейс программы

Функциональные возможности приложения:

– навигациа по месяцам и годам;

При нажатии на кнопку вверху каждого месяца появляется возможность выбрать необходимые месяц (рисунок 2) и год (рисунок 3).

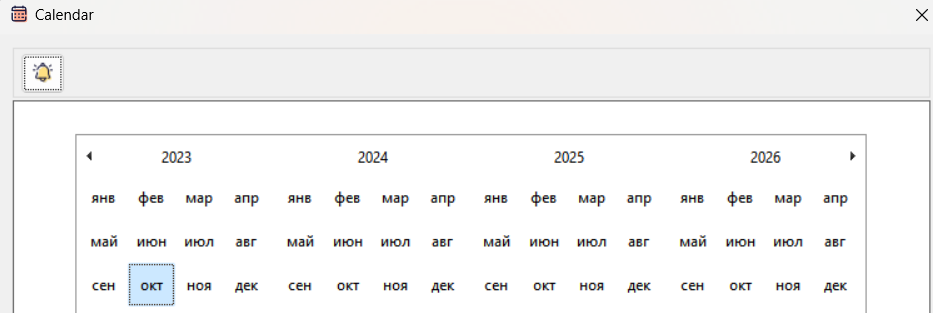


Рисунок 2 – Выбор месяца

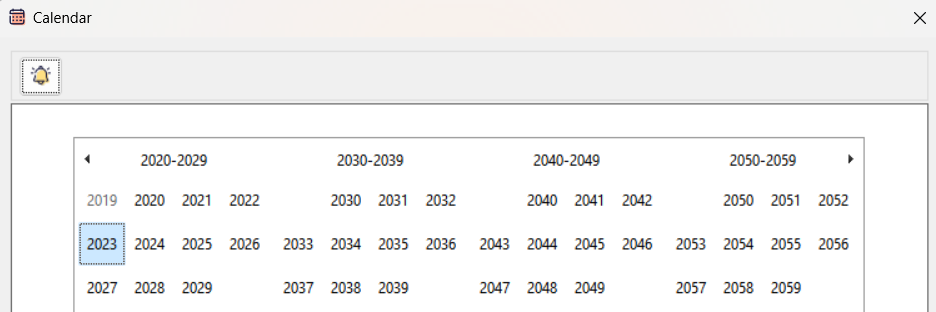


Рисунок 3 – Выбор года

– создание нового события;

При нажатии на кнопку с колокольчиком, открывается диалоговое окно с помощью которого можно добавить новое событие (рисунок 4).

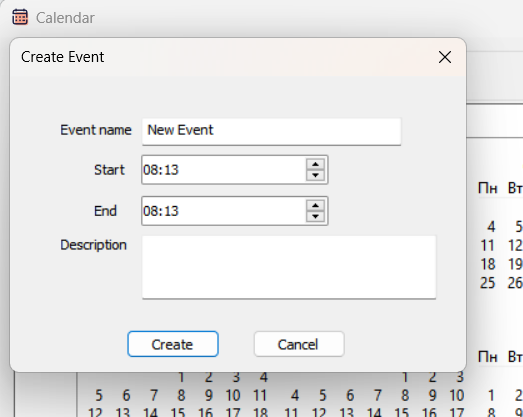


Рисунок 4 – Диалоговое окно для добавления события

– просмотр событий;

При двойном нажатии на дату, появляется диалоговое окно со списком событий, добавленных на выбранную дату (рисунок 5)

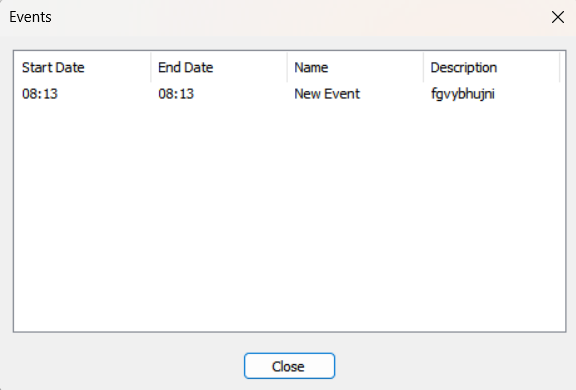


Рисунок 5 – Список событий

– удаление события;

Для удаления необходимо выбрать событие, который надо удалить и нажать на кнопку удаления (рисунок 6, pисунок 7).

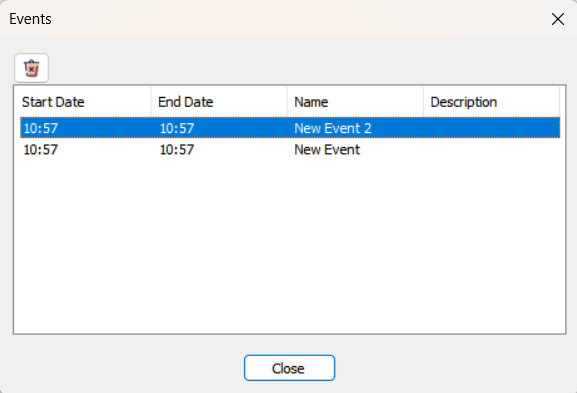


Рисунок 6 – Состояние до удаления события

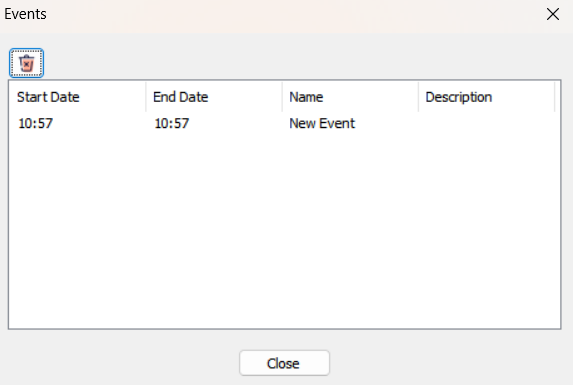


Рисунок 7 – Состояние после удаления события

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Общие сведения об элементах управления [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/controls/common-controls-intro> – Дата доступа 10.10.2023.

[2] About List-View Controls [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/controls/list-view-controls-overview> – Дата доступа 10.10.2023.

[3] Сведения о сообщениях и очередях сообщений [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/winmsg/about-messages-and-message-queues> – Дата доступа 10.10.2023.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл main.cpp**

#include "resource.h"

#include <windows.h>

#include <windowsx.h>

#include <CommCtrl.h>

#include <strsafe.h>

#include <chrono>

#include <string>

#include <map>

#include "Helpers.h"

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

static HINSTANCE hAppInstance = NULL;

static HWND hMainWindow = NULL;

static HWND hCalendarWindow = NULL;

struct Event

{

int StartTimeInMin;

int EndTimeInMin;

std::wstring Name;

std::wstring Description;

};

typedef std::multimap<time\_t, Event> EventContainer;

EventContainer Events;

INT\_PTR CALLBACK MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK AddEventDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

INT\_PTR CALLBACK EventListDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

void OpenAddEventDialog();

void OpenEventListDialog();

int WINAPI wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ PWSTR pCmdLine, \_In\_ int nCmdShow)

{

hAppInstance = hInstance;

INITCOMMONCONTROLSEX iCC = { sizeof(INITCOMMONCONTROLSEX), ICC\_LISTVIEW\_CLASSES | ICC\_DATE\_CLASSES };

::InitCommonControlsEx(&iCC);

hMainWindow = ::CreateDialogParam(hInstance, MAKEINTRESOURCE(IDD\_MAIN\_DIALOG), NULL, MainDialogProc, 0);

if (hMainWindow == NULL)

{

return -1;

}

HICON hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_CALENDAR));

SendMessage(hMainWindow, WM\_SETICON, ICON\_SMALL, (LPARAM)hIcon);

HWND hCalendarParent = ::GetDlgItem(hMainWindow, IDC\_CALENDAR\_STATIC);

hCalendarWindow = CreateWindowEx(0, MONTHCAL\_CLASS, L"", WS\_BORDER | WS\_CHILD | WS\_VISIBLE /\* | MCS\_DAYSTATE\*/,

0, 0, 0, 0, hMainWindow, NULL, hAppInstance, NULL);

if (hCalendarWindow == NULL)

return -1;

MonthCal\_SetCurrentView(hCalendarWindow, MCMV\_MONTH);

RECT rc;

::GetWindowRect(hCalendarParent, &rc);

::MapWindowPoints(HWND\_DESKTOP, hMainWindow, (LPPOINT)&rc, 2);

::SetWindowPos(hCalendarWindow, NULL, rc.left, rc.top - 4, rc.right - rc.left, rc.bottom - rc.top - 4, SWP\_NOZORDER);

::ShowWindow(hMainWindow, nCmdShow);

::UpdateWindow(hMainWindow);

MSG msg = { };

while (::GetMessage(&msg, NULL, 0, 0) > 0)

{

if (!IsDialogMessage(hMainWindow, &msg))

{

::TranslateMessage(&msg);

::DispatchMessage(&msg);

}

}

return 0;

}

INT\_PTR CALLBACK MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY:

::PostQuitMessage(0);

return TRUE;

case WM\_CLOSE:

::DestroyWindow(hwnd);

return TRUE;

case WM\_NOTIFY:

{

static auto lastDaySelectTimeMs = std::chrono::system\_clock::now();

switch (((LPNMHDR)lParam)->code)

{

case MCN\_GETDAYSTATE:

{

NMDAYSTATE\* nmDayState = (NMDAYSTATE\*)lParam;

MONTHDAYSTATE rgMonths[12] = { 0 };

int cMonths = ((NMDAYSTATE\*)lParam)->cDayState;

}

break;

case MCN\_SELECT:

{

if (std::chrono::duration\_cast<std::chrono::milliseconds>(std::chrono::system\_clock::now() - lastDaySelectTimeMs).count() < ::GetDoubleClickTime())

{

OpenEventListDialog();

}

lastDaySelectTimeMs = std::chrono::system\_clock::now();

}

break;

}

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDC\_ADD\_TASK:

{

return TRUE;

}

break;

case IDC\_ADD\_EVENT:

{

OpenAddEventDialog();

return TRUE;

}

break;

}

break;

case WM\_INITDIALOG:

{

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_ADD\_TASK, IDI\_ICON1);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_ADD\_EVENT, IDI\_ADD\_EVENT\_ICON);

}

break;

}

return FALSE;

}

void OpenAddEventDialog()

{

SYSTEMTIME selectedDate;

MonthCal\_GetCurSel(hCalendarWindow, &selectedDate);

time\_t startOfTheDay = Helpers::GetStartOfTheDay(selectedDate);

INT\_PTR result = ::DialogBoxParamW(hAppInstance, MAKEINTRESOURCE(IDD\_ADD\_EVENT\_DIALOG), hMainWindow,

AddEventDialogProc, (LPARAM)startOfTheDay);

if (result == 1)

{

}

}

void OpenEventListDialog()

{

SYSTEMTIME selectedDate;

MonthCal\_GetCurSel(hCalendarWindow, &selectedDate);

time\_t startOfTheDay = Helpers::GetStartOfTheDay(selectedDate);

INT\_PTR result = ::DialogBoxParamW(hAppInstance, MAKEINTRESOURCE(IDD\_LIST\_EVENTS\_DIALOG), hMainWindow,

EventListDialogProc, (LPARAM)startOfTheDay);

if (result == 1)

{

}

}

INT\_PTR CALLBACK AddEventDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

static time\_t CurrentStartOfTheDay = 0;

switch (uMsg)

{

case WM\_CLOSE:

return ::EndDialog(hwnd, 0);

break;

case WM\_INITDIALOG:

{

CurrentStartOfTheDay = (time\_t)lParam;

HWND hStartTime = ::GetDlgItem(hwnd, IDC\_STARTTIME);

DateTime\_SetFormat(hStartTime, L"hh:mm tt");

SYSTEMTIME initStartTime;

Helpers::Time\_tToSystemTime(CurrentStartOfTheDay, &initStartTime);

initStartTime.wHour = 8;

DateTime\_SetSystemtime(hStartTime, GDT\_VALID, &initStartTime);

HWND hEndTime = ::GetDlgItem(hwnd, IDC\_ENDTIME);

DateTime\_SetFormat(hEndTime, L"hh:mm tt");

SYSTEMTIME initEndTime;

Helpers::Time\_tToSystemTime(CurrentStartOfTheDay, &initEndTime);

initEndTime.wHour = 8;

initEndTime.wMinute = 30;

DateTime\_SetSystemtime(hEndTime, GDT\_VALID, &initEndTime);

HWND hEventName = ::GetDlgItem(hwnd, IDC\_EVENT\_NAME);

Edit\_SetText(hEventName, L"New Event");

}

break;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDOK:

{

HWND hStartTime = ::GetDlgItem(hwnd, IDC\_STARTTIME);

SYSTEMTIME startTime;

DateTime\_GetSystemtime(hStartTime, &startTime);

SYSTEMTIME endTime;

HWND hEndTime = ::GetDlgItem(hwnd, IDC\_ENDTIME);

DateTime\_GetSystemtime(hEndTime, &endTime);

HWND hEventName = ::GetDlgItem(hwnd, IDC\_EVENT\_NAME);

WCHAR eventName[1024];

Edit\_GetText(hEventName, eventName, 1024);

HWND hEventDescription = ::GetDlgItem(hwnd, IDC\_EVENT\_DESCRIPTION);

WCHAR eventDescription[1024];

Edit\_GetText(hEventDescription, eventDescription, 1024);

Event newEvent;

newEvent.Name = eventName;

newEvent.Description = eventDescription;

newEvent.StartTimeInMin = startTime.wMinute + startTime.wHour \* 60;

newEvent.EndTimeInMin = endTime.wMinute + endTime.wHour \* 60;

Events.insert(EventContainer::value\_type(CurrentStartOfTheDay, newEvent));

return ::EndDialog(hwnd, 1);

}

break;

case IDCANCEL:

{

return ::EndDialog(hwnd, 0);

}

break;

}

break;

}

}

return FALSE;

}

INT\_PTR CALLBACK EventListDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_CLOSE:

return ::EndDialog(hwnd, 0);

break;

case WM\_INITDIALOG:

{

auto startOfTheDay = (time\_t)lParam;

LVCOLUMNW lvc;

lvc.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

lvc.cx = 120;

lvc.fmt = LVCFMT\_LEFT;

static WCHAR szEventListColumnString[4][64] = { L"Start Date", L"End Date", L"Name", L"Description" };

HWND hEventListView = ::GetDlgItem(hwnd, IDC\_EVENT\_LIST);

ListView\_SetExtendedListViewStyle(hEventListView, LVS\_EX\_AUTOSIZECOLUMNS | LVS\_EX\_FULLROWSELECT);

for (auto column = 0; column < 4; ++column)

{

lvc.pszText = szEventListColumnString[column];

if (ListView\_InsertColumn(hEventListView, column, &lvc) == -1)

{

return FALSE;

}

}

auto eventList = Events.equal\_range(startOfTheDay);

for (auto it = eventList.first; it != eventList.second; ++it)

{

LVITEMW lvi = { };

lvi.mask = LVIF\_TEXT;

ListView\_InsertItem(hEventListView, &lvi);

WCHAR startTimeString[16];

wsprintfW(startTimeString, L"%02d:%02d", it->second.StartTimeInMin / 60, it->second.StartTimeInMin % 60);

ListView\_SetItemText(hEventListView, lvi.iItem, 0, startTimeString);

WCHAR endTimeString[16];

wsprintfW(endTimeString, L"%02d:%02d", it->second.EndTimeInMin / 60, it->second.EndTimeInMin % 60);

ListView\_SetItemText(hEventListView, lvi.iItem, 1, endTimeString);

ListView\_SetItemText(hEventListView, lvi.iItem, 2, &it->second.Name[0]);

ListView\_SetItemText(hEventListView, lvi.iItem, 3, &it->second.Description[0]);

}

}

break;

case WM\_COMMAND:

{

switch (LOWORD(wParam))

{

case IDOK:

{

return ::EndDialog(hwnd, 0);

}

break;

}

}

break;

}

return FALSE;

}

**Файл Helpers.h**

#pragma once

#include <windows.h>

#include <string>

namespace Helpers

{

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID);

void SetControlEnabled(HWND hDlg, int controlID, bool enabled);

std::wstring FileTimeToString(FILETIME ftime);

time\_t GetStartOfTheDay(SYSTEMTIME st);

void Time\_tToSystemTime(time\_t t, SYSTEMTIME\* st);

}

**Файл Helpers.cpp**

#include "Helpers.h"

#include <codecvt>

#include <shlobj.h>

#include <sstream>

#include <iomanip>

#include <ctime>

namespace Helpers

{

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID)

{

HICON hIcon = static\_cast<HICON>(::LoadImage(hInstance, MAKEINTRESOURCE(iconID), IMAGE\_ICON,

16, 16, LR\_DEFAULTCOLOR));

if (hIcon == NULL)

{

return FALSE;

}

::SendDlgItemMessage(hDlg, buttonID, BM\_SETIMAGE, (WPARAM)IMAGE\_ICON, (LPARAM)hIcon);

return TRUE;

}

void SetControlEnabled(HWND hDlg, int controlID, bool enabled)

{

:: EnableWindow(GetDlgItem(hDlg, controlID), enabled);

}

std::wstring FileTimeToString(FILETIME ftime)

{

SYSTEMTIME utc;

::FileTimeToSystemTime(std::addressof(ftime), std::addressof(utc));

std::wostringstream os;

os << std::setfill<wchar\_t>('0') << std::setw(4) << utc.wYear << '-' << std::setw(2) << utc.wMonth

<< '-' << std::setw(2) << utc.wDay << ' ' << std::setw(2) << utc.wHour

<< ':' << std::setw(2) << utc.wMinute << ':' << std::setw(2) << utc.wSecond;

return os.str();

}

time\_t GetStartOfTheDay(SYSTEMTIME st)

{

std::tm tmDate;

tmDate.tm\_sec = 0;

tmDate.tm\_min = 0;

tmDate.tm\_hour = 0;

tmDate.tm\_mday = st.wDay;

tmDate.tm\_mon = st.wMonth - 1;

tmDate.tm\_year = st.wYear - 1900;

tmDate.tm\_isdst = 0;

std::time\_t t = std::mktime(&tmDate);

return t;

}

void Time\_tToSystemTime(time\_t t, SYSTEMTIME\* st)

{

std::tm tm;

localtime\_s(&tm, &t);

st->wYear = tm.tm\_year + 1900;

st->wDay = tm.tm\_mday;

st->wDayOfWeek = tm.tm\_wday;

st->wMonth = tm.tm\_mon + 1;

st->wHour = tm.tm\_hour;

st->wMinute = tm.tm\_min;

st->wSecond = tm.tm\_sec;

}

}