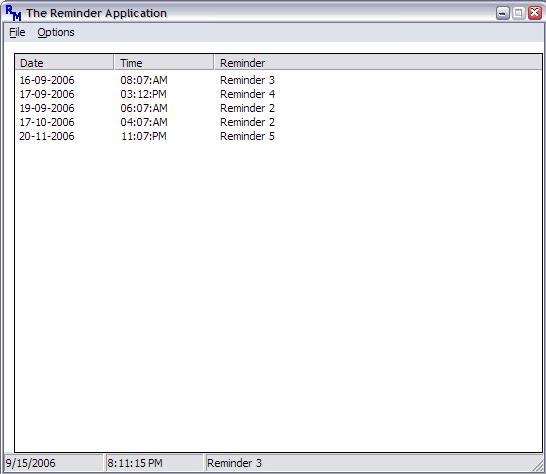
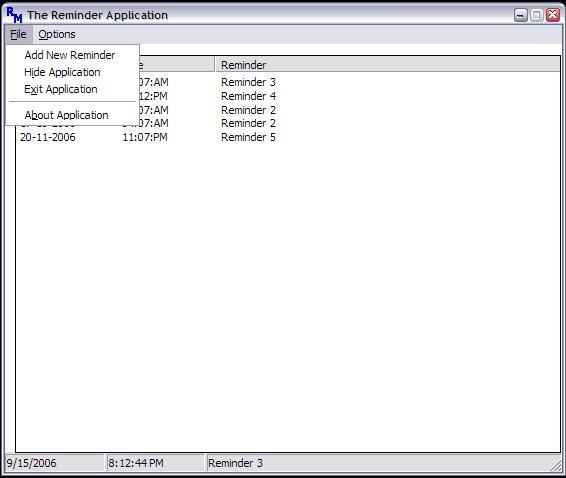
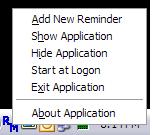
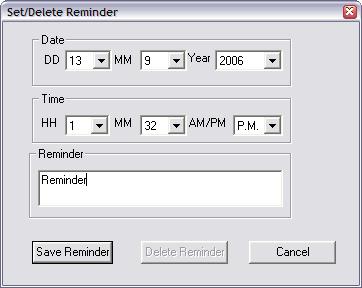
Simple Reminder Application









Introduction

The aim of this project is to develop a reminder application which can store multiple reminders and will display a message box at the time specified by the user. The application can be made to start with Windows, or the user can manually start it. The application menu provides the option for this. The application uses the Windows registry to store the reminder information. I have created a few simple APIs which can be used directly in other applications like for adding, searching data to registry, adding data to a list-view control etc. I have used mouse click notification on the list-view which will read all the information of the particular row where the mouse is clicked. I have added System tray notification and a Status bar to the code as well. Mainly, the work is to manage the registry and search for reminders in a list-view control. I have also implemented sorting in the list-view control based on the date/time of reminders, and the most recent reminder which will be triggered is displayed in the status bar.

I have also trapped the WM\_CLOSE message so that when the user clicks the Close button, the application minimizes and goes to the system tray. This is to keep the application running in the background. Also, I have trapped the minimize button so that minimizing will also put the application in the system tray instead of on the taskbar. To close the application, I have provided a menu option in that.

The application gives the user the option of starting it at Windows start up by putting a registry entry. Right now, the reminder is displayed in a dialog-box; I will try to improve the user-interface over a period of time.

I have used Microsoft Visual Studio 2005 Express edition in this project. For compiling this application in Visual Studio 6.0, I would suggest starting an empty Win32 application (not console) and copying all the files in the project and compiling it. Please add these libraries in the linker option of the project: *comctl32.lib* and *Psapi.lib*.

Using the code

Here, I will show some of the code snippets which have been incorporated in the code.

The following code checks if application is already running. If yes, then it brings the window in focus, else starts a new instance of the program.

Hide   Copy Code

*// szAppName is application name*

*// //to see if program is running. if yes it*

*// activates it and exits the second instance*

hwndPrevInstance=FindWindow(szAppName,NULL);

if(hwndPrevInstance)

{

*//if eliminated it gets default value from*

*//winmain.This initialization will set window*

*//to normal and active positon*

iCmdShow=SW\_SHOWNORMAL;

ShowWindow(hwndPrevInstance,SW\_SHOWNORMAL);

return 0;

}

*//*

The window is always placed in the centre of the screen by calling the user-defined API: BOOL CenterWindow(HWND hwnd).

Hide   Copy Code

*// BOOL CenterWindow(HWND hwnd)*

{

HWND hwndParent;

RECT rect, rectP;

int width, height;

int screenwidth, screenheight;

int x, y;

*//make the window relative to its desktop*

hwndParent = GetDesktopWindow();

GetWindowRect(hwnd, &rect);

GetWindowRect(hwndParent, &rectP);

width = rect.right - rect.left;

height = rect.bottom - rect.top;

x = ((rectP.right-rectP.left) - width) / 2 + rectP.left;

y = ((rectP.bottom-rectP.top) - height) / 2 + rectP.top;

screenwidth = GetSystemMetrics(SM\_CXSCREEN);

screenheight = GetSystemMetrics(SM\_CYSCREEN);

*//make sure that the dialog box never moves outside of*

*//the screen*

if(x < 0) x = 0;

if(y < 0) y = 0;

if(x + width > screenwidth) x = screenwidth - width;

if(y + height > screenheight) y = screenheight - height;

MoveWindow(hwnd, x, y, width, height, FALSE);

return TRUE;

}

The code detects the path from where the application is started and sees if the option of running the application with Windows is set. This is done to produce a check mark of if the status is yes in the menu. The menu is created at runtime using Win32 APIs instead of a normal resource file.

Hide   Copy Code

*// code in WM\_CREATE*

*//open process to get application name*

hProcess=OpenProcess(PROCESS\_QUERY\_INFORMATION|

PROCESS\_VM\_READ, FALSE,

GetCurrentProcessId());

*//Get directory from where exe has started*

*//and reserve 100 TCHARs for AppName*

szApplicationPathnName=(TCHAR\*)malloc(sizeof(TCHAR)\*

(GetCurrentDirectory(0,NULL)+100));

*//get the full path including the application Name*

GetModuleFileNameEx(hProcess,NULL,szApplicationPathnName,

lstrlen(szApplicationPathnName));

*//display menu bar*

hMenu=MenuDisplay(hwnd);

*//check start application at windows start up after menu is created*

*//so that option can be checked/unchecked*

CheckApplicationStartupWithWindows(hwnd, hMenu);

The application runs two timers, one for updating the time/date in the application, and the other to check if any reminder time matches with the system time. The CreateTrayIcon(hwnd) creates a system tray icon, andCreateListViewBox(::hInstance,hwnd) creates a list-view control, and CreateApplicationKey()creates/opens the registry key for the application.

Hide   Copy Code

*//timer for status bar time update*

SetTimer(hwnd,ID\_TIMER\_1,950,TimerProc);

*//timer for every minute registry comparison with current time*

SetTimer(hwnd,ID\_TIMER\_2,58000,TimerProc);

*//create tray icon*

CreateTrayIcon(hwnd);

*//create status bar and store the handle*

hStatusBar = CreateStatusBar(hwnd);

*//create listview control and store the handle*

hListViewControl = CreateListViewBox(::hInstance,hwnd);

*//main registry key in which furthe data will be stored*

hApplicationKey=CreateApplicationKey();

*//*

Once the reminder is triggered, it is automatically deleted from the registry as well the list-view control.

To add or remove reminders, double click on the list-view control or select “Add New Reminder” from the menu (both menu-bar and system tray). A dialog box will open where the user can feed the data. If it is double clicked on any reminder, then it can be used to either edit or delete the reminder. The code below was implemented in the main Windows callback function. And if there is data where mouse is clicked, then the data is stored in theCLICKEDITEMDATA structure which will store all the information like date/time/reminder, and also a flag which shows if data was there or not. This flag will have 0 to indicate no data, and 1 to show data is present. This is useful to fill the pop-up dialog which is displayed with all the relevant data in it.

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

*// case WM\_NOTIFY:*

{

NMITEMACTIVATE \*nmhdr=NULL;

nmhdr=(NMITEMACTIVATE \*)lParam;

int itemclicked;

LRESULT i;

LVHITTESTINFO pinfo;

LVITEM LvItem={sizeof(LVITEM)};

if(hListViewControl==nmhdr->hdr.hwndFrom)

{

switch(nmhdr->hdr.code)

{

case NM\_DBLCLK:

*//when double clicked item info comes*

*//in NMITEMACTIVATE structure*

*//itemclicked is zero based index*

*//if its clicked on subitem then we*

*//need to send LVM\_SUBITEMHITTEST message*

*//with mouse coordiantes filled*

*//in point structure of LVHITTESTINFO*

pinfo.pt.x=nmhdr->ptAction.x;

pinfo.pt.y=nmhdr->ptAction.y;

i = SendMessage(hListViewControl,

(UINT) LVM\_SUBITEMHITTEST ,

0,(LPARAM) &pinfo );

itemclicked = pinfo.iItem;

if(-1==itemclicked)

{

ClickedItemData.DatainsideListBox=0;

*//no item on list box*

}

else

{

ClickedItemData.DatainsideListBox=1;

LvItem.mask=LVIF\_TEXT;

LvItem.iItem=itemclicked;

LvItem.cchTextMax=sizeof(ClickedItemData.szDate);

LvItem.pszText=ClickedItemData.szDate;

i=SendMessage(hListViewControl,

LVM\_GETITEM,0,(LPARAM)&LvItem);

LvItem.iSubItem=1;

LvItem.cchTextMax=sizeof(ClickedItemData.szTime);

LvItem.pszText=ClickedItemData.szTime;

i=SendMessage(hListViewControl,

LVM\_GETITEM,0,(LPARAM)&LvItem);

LvItem.iSubItem=2;

LvItem.cchTextMax=sizeof(ClickedItemData.szReminder);

LvItem.pszText=ClickedItemData.szReminder;

i=SendMessage(hListViewControl,

LVM\_GETITEM,0,

(LPARAM)&LvItem);

}

*//here create a dialog box*

DialogBoxW(::hInstance, MAKEINTRESOURCE(IDD\_DIALOG1),

hwnd,DialogProc);

return 0;

case NM\_CLICK:

*//Beep(100,500);*

return 0;

case NM\_RCLICK:

*// Beep(100,500);*

return 0;

default:

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

}

}

else

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

} *//*

Points of interest

I have come up with some APIs which can be very useful in other applications. I will mention all of them with a small comment.

**Registry operations**

* HKEY CreateApplicationKey() - Creates a main registry key with the application name.
* int AddDatatoRegistry(const HKEY hKey, TCHAR \*szDate,TCHAR \*szTime,TCHAR \*szReminder) - a general function to add registry data with a Key(szDate),KeyValue(szTime) andKeyValueData(szReminder).
* int DeleteDatafromRegistry(const HKEY hKey, TCHAR \*szDate,TCHAR \*szTime) - the general function to delete registry data with a Key(szDate),KeyValue(szTime). TheKeyValue(szTime) gets deleted along with the data inside.
* int SearchRegistryString(const HKEY hKey,TCHAR \*szSearchDate,TCHAR \*szSearchTime,TCHAR \*szReminderData) - search subkey and value name in registry; if found, return the value data.
* UpdateListViewfromRegistry(const HWND hListView,const HKEY hKey) - numerate all the data stored in registry and then add it to listview.
* int DeleteEmptyKeysfromRegistry(const HKEY hKey) - this searches for any empty keys stored, and deletes so that the registry doesn't exceed necessarily.

Menu options

* HMENU MenuDisplay(const HWND hwnd) - create a menu for the application without using resources.
* HMENU SystemTrayMenu(const HWND hwnd) - create a pop-up menu when clicked on the tray icon.

**Status Bar**

* HWND CreateStatusBar(const HWND hwnd) - create the Status bar.
* void DisplayClockDateStatusBar(const HWND hStatusBar) - Display time in parts of the status bar.

**ListView control**

* HWND CreateListViewBox(const HINSTANCE hInstance,const HWND hwnd) - create the ListView control box.
* int AddListViewRow(const HWND hControl, TCHAR \*szItemText, TCHAR \*szSubItemText1, TCHAR \*szSubItemText2) - a general function to fill the list view with all three parameters.
* int CALLBACK SortingCodeListViewControl(LPARAM lParam1, LPARAM lParam2,LPARAM lParamSort) – a function for sorting items in the list-view based on time/date values.

**General APIs which were used in code to perform some specific tasks**

* void DateStringtoDateComponents(TCHAR \*szDate,short \*day, short \*month, short\*year) - convert a date string to integer values of day, month, and year, used for filling data in the dialog box.
* void DateComponentstoDateString(const short day,const short month,const shortyear,TCHAR \*szDateString) - to convert day, month, and year in the date string, which can be stored in registry and displayed in a listview control.
* void TimeStringtoTimeComponents(TCHAR \*szTime,short \*hour, short \*minute, short\*AMP) - convert a time string to integer values of hour, minute, and AM/PM, used for filling data in a dialog box.
* void TimeComponentstoTimeString(const short hour,const short minute,const shortAMP,TCHAR \*szTimeString) - to convert hour, minute, and AM/PM in the time string, which can be stored in the registry and displayed in the listview control.
* int FillDefaultDatainDialogBox(const HWND hDlg) - initialise the drop down values in the various input parameters in the dialog box.
* int SaveDialogButton(const HWND hDlg,const HKEY hKey) - for the Save button clicked on the dialog.
* int DeleteDialogButton(const HWND hDlg,const HKEY hKey) - for the Delete button on the dialog box.

To minimize the application when the minimize button is clicked:

Hide   Copy Code

case WM\_ACTIVATE:

if(HIWORD(wParam)!=0&&LOWORD(wParam)==WA\_INACTIVE)

ShowWindow(hwnd,SW\_HIDE);

return 0;