```
/*
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
////////////////////////////////////////////////////////////
//  ACLLib - Advanced C Lab Library
//    Ver. 2014-07
//For Students' Lab at Zhejiang University
//Created 2008 by Gao Yuan
//Modified 2009 by Cui Liwei
//    2010 by Lan Huidong
//Revised2012 by Li Rui
//  Modified  2014 by Weng Kai for MOOC
////////////////////////////////////////////////////////////
#define _CRT_SECURE_NO_WARNINGS
#define _CRT_NON_CONFORMING_SWPRINTFS
#define CINTERFACE
#ifdef _UNICODE
#undef _UNICODE
#endif
#ifdef UNICODE
#undef UNICODE
#endif
#include "acllib.h"
#include <windows.h>
#include <olectl.h>
#include <stdio.h>
#ifdef _MSC_VER
#pragma comment(lib,"winmm.lib")
#pragma comment(lib,"msimg32.lib")
#endif
#ifdef _DEBUG
#define ACL_ASSERT(_Expression,errStr) (void)( (!!(_Expression)) || (acl_error(errStr),0) )
#else
#define ACL_ASSERT(flag,errStr) ((void)0)
#endif
#define ACL_ASSERT_HWND ACL_ASSERT(g_hWnd!=0, \
"You should call function \"initWindow(...)\" befor use function \"" __FUNCTION__ "\"" )
#define ACL_ASSERT_BEGIN_PAINT ACL_ASSERT(g_hmemdc!=0, \
"You should call function \"beginPaint()\" befor use function \"" __FUNCTION__ "\"" )
// f
int Setup(void);
const char g_wndClassName[] = "ACL_WND_CLASS";
const char g_libName[] = "ACLLIB";
HINSTANCE g_hInstance;
HWND g_hWnd = NULL;
HDC g_hmemdc = NULL;
HBITMAP g_hbitmap = NULL;
int g_wndHeight;
int g_wndWidth;
HPEN g_pen = NULL;
ACL_Color g_penColor = BLACK;
int g_penWidth = 1;
int g_penStyle = PEN_STYLE_SOLID;
HBRUSH g_brush = NULL;
ACL_Color g_brushColor = BLACK;
int g_brushStyle = BRUSH_STYLE_SOLID;
HFONT g_font = NULL;
char g_fontName[256] = "ËÎÌå";
int g_textSize = 12;
ACL_Color g_textColor = BLACK;
ACL_Color g_textBkColor = WHITE;
int g_caretHeight = 12;
int g_caretWidth = 6;
int g_caretX = 0;
int g_caretY = 0;
int g_soundID = 0;
KeyboardEventCallback g_keyboard = NULL;
MouseEventCallback g_mouse = NULL;
TimerEventCallback g_timer = NULL;
CharEventCallback g_char = NULL;
```

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
//
void acl_error(char *errStr)
{
MessageBoxA(g_hWnd,errStr,g_libName,MB_ICONERROR);
exit(0);
}
//
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
{
MSG          msg;
WNDCLASSA     wndclass;

g_hInstance = hInstance;
g_hWnd = NULL;
    g_keyboard = NULL;
g_mouse = NULL;
g_timer = NULL;
wndclass.style         = CS_HREDRAW | CS_VREDRAW | CS_OWNDC | CS_DBLCLKS;
wndclass.lpfnWndProc   = WndProc;
wndclass.cbClsExtra    = 0;
wndclass.cbWndExtra    = 0;
wndclass.hInstance     = hInstance;
wndclass.hInstance     = hInstance;
wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
wndclass.hbrBackground = (HBRUSH) GetStockObject (BLACK_BRUSH);
wndclass.lpszMenuName  = NULL;
wndclass.lpszClassName = g_wndClassName;
if (!RegisterClassA(&wndclass))
{
MessageBoxA(NULL, "This program requires Windows NT!", g_libName, MB_ICONERROR);
return 0;
}
Setup();
ACL_ASSERT(g_hWnd,"You must call \"initWindow(...)\" in Main()");
while (GetMessage(&msg, NULL, 0, 0))
{
TranslateMessage (&msg);
DispatchMessage (&msg);
}
return msg.wParam;
}
//
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
switch (message)
{
case WM_CREATE:
{
HDC hdc;
hdc = GetDC(hwnd);
g_hbitmap = CreateCompatibleBitmap(
hdc, GetSystemMetrics(SM_CXSCREEN), GetSystemMetrics(SM_CYSCREEN));
g_hmemdc = CreateCompatibleDC(hdc);
SelectObject(g_hmemdc, g_hbitmap);
BitBlt(g_hmemdc,
0, 0,
GetSystemMetrics(SM_CXSCREEN),
GetSystemMetrics(SM_CYSCREEN),
g_hmemdc,
0, 0,
WHITENESS);
DeleteDC(g_hmemdc);
ReleaseDC(hwnd, hdc);
CreateCaret(hwnd,0,g_caretWidth,g_caretHeight);
g_caretX = g_wndWidth;
g_caretY = g_wndHeight;
SetCaretPos(g_caretX,g_caretY);
break;
}
case WM_ERASEBKGND:
break;
case WM_PAINT:
{
HDC hdc;
PAINTSTRUCT ps;
RECT rect;
hdc = BeginPaint(hwnd, &ps);
g_hmemdc = CreateCompatibleDC(hdc);
SelectObject(g_hmemdc, g_hbitmap);
```

```
GetClientRect(hwnd,&rect);
BitBlt(hdc, 0, 0, rect.right - rect.left,
rect.bottom - rect.top, g_hmemdc, 0, 0, SRCCOPY);
DeleteDC(g_hmemdc);
g_hmemdc = 0;
EndPaint(hwnd,&ps);
break;
}
case WM_CHAR:
if (g_char != NULL)
g_char((char) wParam);
break;
case WM_KEYDOWN:
if (g_keyboard != NULL)
g_keyboard((int) wParam,KEY_DOWN);
break;
case WM_KEYUP:
if(g_keyboard != NULL)
g_keyboard((int) wParam,KEY_UP);
break;
case WM_LBUTTONDOWN:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), LEFT_BUTTON, BUTTON_DOWN);
break;
case WM_LBUTTONUP:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), LEFT_BUTTON, BUTTON_UP);
break;
case WM_LBUTTONDBLCLK:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), LEFT_BUTTON, BUTTON_DOUBLECLICK);
break;
case WM_MBUTTONDOWN:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), MIDDLE_BUTTON, BUTTON_DOWN);
break;
case WM_MBUTTONUP:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), MIDDLE_BUTTON, BUTTON_UP);
break;
case WM_MBUTTONDBLCLK:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), MIDDLE_BUTTON, BUTTON_DOUBLECLICK);
break;
case WM_RBUTTONDOWN:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), RIGHT_BUTTON, BUTTON_DOWN);
break;
case WM_RBUTTONUP:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), RIGHT_BUTTON, BUTTON_UP);
break;
case WM_RBUTTONDBLCLK:
if (g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), RIGHT_BUTTON, BUTTON_DOUBLECLICK);
break;
case WM_MOUSEMOVE:
if(g_mouse != NULL)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam), MOUSEMOVE, MOUSEMOVE);
break;
case WM_MOUSEWHEEL:
if(g_mouse == NULL)
break;
if(HIWORD(wParam) == 120)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam),MIDDLE_BUTTON,ROLL_UP);
else if(HIWORD(wParam)==65416)
g_mouse((int) LOWORD(lParam), (int) HIWORD(lParam),MIDDLE_BUTTON,ROLL_DOWN);
break;
case WM_TIMER:
if (g_timer != NULL)
g_timer(wParam);
break;
case WM_DESTROY:
DeleteObject(g_hbitmap);
PostQuitMessage(0);
break;
default:
return DefWindowProc(hwnd, message, wParam, lParam);
}
return 0;
}
```

```
//
void initWindow(const char *wndName, int x, int y, int width, int height)
{
RECT rect;
ACL_ASSERT(!g_hWnd,"Don't call initWindow twice");
g_wndHeight = height;
g_wndWidth = width;
if(x==DEFAULT || y==DEFAULT)
x=y=CW_USEDEFAULT;
g_hWnd = CreateWindowA (
        g_wndClassName, wndName,
WS_OVERLAPPEDWINDOW & ~WS_MAXIMIZEBOX & ~WS_SIZEBOX,
x, y,
width, height,
NULL, NULL, 0, NULL) ;
    if(!g_hWnd)
    {
MessageBoxA(NULL,"Fail to create window",g_libName,MB_ICONERROR);
        exit(0);
    }
GetClientRect(g_hWnd,&rect);
width += width - (rect.right-rect.left);
height += height - (rect.bottom-rect.top);
SetWindowPos(g_hWnd,HWND_TOP,0,0,width,height,SWP_NOMOVE);
ShowWindow (g_hWnd,1);
UpdateWindow (g_hWnd);
}
void initConsole(void)
{
    AllocConsole();
    freopen("CONIN$", "r+t", stdin);
    freopen("CONOUT$", "w+t", stdout);
}
void msgBox(const char title[],const char text[],int flag)
{
ACL_ASSERT_HWND;
MessageBoxA(g_hWnd,text,title,flag);
}
//
void updatePen();
void updateBrush();
void updateFont();
//
void beginPaint()
{
HDC hdc;
ACL_ASSERT_HWND;
hdc = GetDC(g_hWnd);
g_hmemdc = CreateCompatibleDC(hdc);
SelectObject(g_hmemdc,g_hbitmap);

updatePen();
updateBrush();
updateFont();
setTextColor(g_textColor);
setTextBkColor(g_textBkColor);
}
void endPaint()
{
DeleteDC(g_hmemdc);
g_hmemdc = 0;
    InvalidateRect(g_hWnd,0,0);
DeleteObject(g_pen);
DeleteObject(g_brush);
DeleteObject(g_font);
g_pen = NULL;
g_brush = NULL;
g_font = NULL;
}
void clearDevice(void)
{
ACL_ASSERT_BEGIN_PAINT;
BitBlt(
g_hmemdc,
0, 0,
GetSystemMetrics(SM_CXSCREEN),
GetSystemMetrics(SM_CYSCREEN) ,
g_hmemdc,
0, 0,
WHITENESS);
}
```

```
void updatePen()
{
if(g_pen)DeleteObject(g_pen);
if(g_penColor==EMPTY)
g_pen = (HPEN)GetStockObject(NULL_PEN);
else
g_pen = CreatePen(g_penStyle,g_penWidth,g_penColor);
SelectObject(g_hmemdc,g_pen);
}
void updateBrush()
{
if(g_brush)DeleteObject(g_brush);
if(g_brushColor==EMPTY)
{
g_brush = (HBRUSH)GetStockObject(NULL_BRUSH);
}
else
{
if(g_brushStyle==BRUSH_STYLE_SOLID)
g_brush = CreateSolidBrush(g_brushColor);
else
g_brush = CreateHatchBrush(g_brushStyle,g_brushColor);
}
SelectObject(g_hmemdc,g_brush);
}
void updateFont()
{
if(g_font)DeleteObject(g_font);
g_font = CreateFontA(
g_textSize,
0,
0,0,700,0,0,0,0,0,0,0,g_fontName);
SelectObject(g_hmemdc,g_font);
}
void setPenColor(ACL_Color newColor)
{
ACL_ASSERT_BEGIN_PAINT;
    g_penColor = newColor;
    updatePen();
}
void setPenWidth(int width)
{
ACL_ASSERT_BEGIN_PAINT;
g_penWidth = width;
updatePen();
}
void setPenStyle(ACL_Pen_Style newStyle)
{
ACL_ASSERT_BEGIN_PAINT;
switch(newStyle)
{
case PEN_STYLE_SOLID:
g_penStyle = PS_SOLID; break;
case PEN_STYLE_DASH:
g_penStyle = PS_DASH; break;
case PEN_STYLE_DOT:
g_penStyle = PS_DOT; break;
case PEN_STYLE_DASHDOT:
g_penStyle = PS_DASHDOT; break;
case PEN_STYLE_DASHDOTDOT:
g_penStyle = PS_DASHDOTDOT; break;
case PEN_STYLE_NULL:
g_penStyle = -1;
setPenColor(EMPTY);
return;
default:
break;
}
updatePen();
}
void setBrushColor(ACL_Color newColor)
{
ACL_ASSERT_BEGIN_PAINT;
    g_brushColor = newColor;
    updateBrush();
}
void setBrushStyle(ACL_Brush_Style newStyle)
{
ACL_ASSERT_BEGIN_PAINT;
switch(newStyle)
{
```

```
case BRUSH_STYLE_SOLID:
g_brushStyle = BRUSH_STYLE_SOLID; break;
case BRUSH_STYLE_HORIZONTAL:
g_brushStyle = HS_HORIZONTAL; break;
case BRUSH_STYLE_VERTICAL:
g_brushStyle = HS_VERTICAL; break;
case BRUSH_STYLE_FDIAGONAL:
g_brushStyle = HS_FDIAGONAL; break;
case BRUSH_STYLE_BDIAGONAL:
g_brushStyle = HS_BDIAGONAL; break;
case BRUSH_STYLE_CROSS:
g_brushStyle = HS_CROSS; break;
case BRUSH_STYLE_DIAGCROSS:
g_brushStyle = HS_DIAGCROSS; break;
case BRUSH_STYLE_NULL:
g_brushStyle = BRUSH_STYLE_SOLID;
setBrushColor(EMPTY);
return;
default:
break;
}
updateBrush();
}
void setTextColor(ACL_Color color)
{
ACL_ASSERT_BEGIN_PAINT;
ACL_ASSERT(color!=EMPTY,"text color can not be EMPTY");
    g_textColor = color;
SetTextColor(g_hmemdc,color);
}
void setTextBkColor(ACL_Color color)
{
ACL_ASSERT_BEGIN_PAINT;
    g_textBkColor = color;
if(color == EMPTY)
SetBkMode(g_hmemdc,TRANSPARENT);
else
{
SetBkMode(g_hmemdc,OPAQUE);
SetBkColor(g_hmemdc,color);
}
}
void setTextSize(int size)
{
ACL_ASSERT_BEGIN_PAINT;
g_textSize = size;
updateFont();
}
void setTextFont(const char *pfn)
{
size_t len;
ACL_ASSERT_BEGIN_PAINT;
len = strlen(pfn);
strcpy(g_fontName,pfn);
updateFont();
}
void paintText(int x, int y, const char *textstring)
{
ACL_ASSERT_BEGIN_PAINT;
TextOutA(g_hmemdc, x, y, textstring, strlen(textstring));
}
void putPixel(int x, int y, ACL_Color color)
{
ACL_ASSERT_BEGIN_PAINT;
SetPixel(g_hmemdc, x, y, color);
}
ACL_Color getPixel(int x, int y)
{
ACL_ASSERT_BEGIN_PAINT;
return GetPixel(g_hmemdc, x, y);
}
int getWidth(void)
{
RECT rect;
GetClientRect(g_hWnd, &rect);
return rect.right;
}
int getHeight(void)
{
RECT rect;
GetClientRect(g_hWnd, &rect);
```

```
return rect.bottom;
}
int getX(void)
{
POINT point;
ACL_ASSERT_BEGIN_PAINT;
GetCurrentPositionEx(g_hmemdc, &point);
return (int) point.x;
}
int getY(void)
{
POINT point;
ACL_ASSERT_BEGIN_PAINT;
GetCurrentPositionEx(g_hmemdc, &point);
return (int) point.y;
}
void moveTo(int x, int y)
{
ACL_ASSERT_BEGIN_PAINT;
MoveToEx(g_hmemdc, x, y,NULL);
}
void moveRel(int dx, int dy)
{
POINT point;
ACL_ASSERT_BEGIN_PAINT;
GetCurrentPositionEx(g_hmemdc, &point);
MoveToEx(g_hmemdc, (int) point.x + dx, (int) point.y + dy,NULL);
}
// Lines and Curves
void arc(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4)
{
ACL_ASSERT_BEGIN_PAINT;
Arc(g_hmemdc,x1,y1,x2,y2,x3,y3,x4,y4);
}
void line(int x0, int y0, int x1, int y1)
{
POINT point;
ACL_ASSERT_BEGIN_PAINT;
GetCurrentPositionEx(g_hmemdc, &point);
MoveToEx(g_hmemdc, x0, y0, NULL);
LineTo(g_hmemdc, x1, y1);
MoveToEx(g_hmemdc,point.x,point.y,NULL);
}
void lineTo(int x, int y)
{
ACL_ASSERT_BEGIN_PAINT;
LineTo(g_hmemdc, x, y);
}
void lineRel(int dx, int dy)
{
POINT point;
ACL_ASSERT_BEGIN_PAINT;
GetCurrentPositionEx(g_hmemdc, &point);
LineTo(g_hmemdc, (int) point.x + dx, (int) point.y + dy);
}
void polyBezier(const POINT *lppt,int cPoints)
{
PolyBezier(g_hmemdc,lppt,cPoints);
}
void polyLine(const POINT *lppt, int cPoints)
{
Polyline(g_hmemdc,lppt,cPoints);
}
// Filled Shapes
void chrod(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)
{
ACL_ASSERT_BEGIN_PAINT;
Chord(g_hmemdc,x1, y1, x2, y2, x3, y3, x4, y4);
}
void ellipse(int left,int top,int right, int bottom)
{
ACL_ASSERT_BEGIN_PAINT;
Ellipse(g_hmemdc,left,top,right,bottom);
}
void pie(int left, int top, int right, int bottom, int xr1, int yr1, int xr2, int yr2)
{
ACL_ASSERT_BEGIN_PAINT;
Pie(g_hmemdc,left,top,right,bottom,xr1,yr1,xr2,yr2);
}
void polygon(const POINT *apt,int cpt)
{
```

```
ACL_ASSERT_BEGIN_PAINT;
Polygon(g_hmemdc,apt,cpt);
}
void rectangle(int left,int top,int right,int bottom)
{
ACL_ASSERT_BEGIN_PAINT;
Rectangle(g_hmemdc,left,top,right,bottom);
}
void roundrect(int left,int top,int right,int bottom,int width,int height)
{
ACL_ASSERT_BEGIN_PAINT;
RoundRect(g_hmemdc,left,top,right,bottom,width,height);
}
void polyline(POINT *apt,int cpt)
{
ACL_ASSERT_BEGIN_PAINT;
Polyline(g_hmemdc,apt,cpt);
}
void putImage(ACL_Image *pImage, int x, int y)
{
HDC hbitmapdc;
ACL_ASSERT_BEGIN_PAINT;
hbitmapdc = CreateCompatibleDC(g_hmemdc);
SelectObject(hbitmapdc, pImage->hbitmap);
BitBlt(g_hmemdc, x, y, pImage->width, pImage->height, hbitmapdc,0,0,SRCCOPY);
DeleteDC(hbitmapdc);
}
void putImageScale(ACL_Image *pImage,int x,int y,int width,int height)
{
HDC hbitmapdc;
ACL_ASSERT_BEGIN_PAINT;
hbitmapdc = CreateCompatibleDC(g_hmemdc);
SelectObject(hbitmapdc, pImage->hbitmap);
if(width == -1)width = pImage->width;
if(height == -1)height = pImage->height;
SetStretchBltMode(g_hmemdc,COLORONCOLOR);
StretchBlt( g_hmemdc,x,y,width,height,hbitmapdc,0,0,pImage->width,pImage->height,SRCCOPY);
DeleteDC(hbitmapdc);
}
void putImageTransparent(ACL_Image *pImage,int x,int y,int width,int height, ACL_Color bkColor)
{
HDC hbitmapdc;
ACL_ASSERT_BEGIN_PAINT;
hbitmapdc = CreateCompatibleDC(g_hmemdc);
SelectObject(hbitmapdc, pImage->hbitmap);
if(width == -1)width = pImage->width;
if(height == -1)height = pImage->height;
//SetStretchBltMode(g_hmemdc,COLORONCOLOR);
TransparentBlt(g_hmemdc,x,y,width,height,hbitmapdc,0,0,pImage->width,pImage->height,bkColor);
DeleteDC(hbitmapdc);
}
void loadImage(const char *image, ACL_Image *mapbuf)
{
HDC hmapdc;
IPicture *ipicture;
IStream *istream;
DWORD filesize = 0, bytes;
OLE_XSIZE_HIMETRIC width;
OLE_YSIZE_HIMETRIC height;
HANDLE file = NULL;
HGLOBAL global = NULL;
LPVOID data = NULL;
ACL_ASSERT_HWND;
file = CreateFileA(image, GENERIC_READ,FILE_SHARE_READ,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL, NULL);
if(file == INVALID_HANDLE_VALUE)
acl_error("Fail to load image, File not exist");
filesize = GetFileSize(file, NULL);
global = GlobalAlloc(GMEM_MOVEABLE, filesize);
data = GlobalLock(global);
ReadFile(file, data, filesize, &bytes, NULL);
GlobalUnlock(global);
CreateStreamOnHGlobal(global, TRUE, &istream);
OleLoadPicture(istream, filesize, TRUE, &IID_IPicture, (LPVOID*)&ipicture);
ipicture->lpVtbl->get_Width(ipicture, &width);
ipicture->lpVtbl->get_Height(ipicture, &height);
mapbuf->width = (int)(width / 26.45833333333);
mapbuf->height = (int)(height / 26.45833333333);
hmapdc = CreateCompatibleDC(GetDC(g_hWnd));
if (mapbuf->hbitmap != NULL)
DeleteObject(mapbuf->hbitmap);
mapbuf->hbitmap = CreateCompatibleBitmap(GetDC(g_hWnd), mapbuf->width, mapbuf->height);
```

```
SelectObject(hmapdc, mapbuf->hbitmap);
    ipicture->lpVtbl->Render(ipicture, hmapdc, 0, 0, mapbuf->width, mapbuf->height, 0, height, width, -
height, NULL);
ipicture->lpVtbl->Release(ipicture);
istream->lpVtbl->Release(istream);
DeleteDC(hmapdc);
GlobalFree(global);
CloseHandle(file);
}
void freeImage(ACL_Image *mapbuf)
{
if(mapbuf->hbitmap) return;
DeleteObject(mapbuf->hbitmap);
mapbuf->hbitmap = NULL;
}
void registerKeyboardEvent(KeyboardEventCallback callback)
{
g_keyboard = callback;
}
void registerCharEvent(CharEventCallback callback)
{
g_char = callback;
}
void registerMouseEvent(MouseEventCallback callback)
{
g_mouse = callback;
}
void registerTimerEvent(TimerEventCallback callback)
{
g_timer = callback;
}
void startTimer(int id,int timeinterval)
{
SetTimer(g_hWnd, id, timeinterval, NULL);
}
void cancelTimer(int id)
{
KillTimer(g_hWnd, id);
}
void loadSound(const char *fileName,ACL_Sound *pSound)
{
char *cmdStr;
int len = strlen(fileName)*sizeof(char);
len +=64;
cmdStr = (char*)malloc(len);
sprintf(cmdStr,"open \"%s\" type mpegvideo alias S%d",fileName,g_soundID);
*pSound = g_soundID;
++g_soundID;
mciSendStringA(cmdStr,NULL,0,NULL);
free(cmdStr);
}
void playSound(int sid,int repeat)
{
char cmdStr[32];
stopSound(sid);
if(repeat)
sprintf(cmdStr,"play S%d from 0 repeat",sid);
else
sprintf(cmdStr,"play S%d from 0",sid);
mciSendStringA(cmdStr,NULL,0,NULL);
}
void stopSound(int sid)
{
char cmdStr[32];
sprintf(cmdStr,"stop S%d",sid);
mciSendStringA(cmdStr,NULL,0,NULL);
}
void setCaretSize(int w,int h)
{
DestroyCaret();
CreateCaret(g_hWnd,0,w,h);
SetCaretPos(g_caretX,g_caretY);
}
void setCaretPos(int x,int y)
{
g_caretX = x;
g_caretY = y;
SetCaretPos(g_caretX,g_caretY);
}
void showCaret()
{
```

```
ShowCaret(g_hWnd);
}
void hideCaret()
{
HideCaret(g_hWnd);
}
```