**C Language Project Report**

**Project Name: GDS Music Player**

**Group Number: 11**

**Group Members: 季陈石(生物方向, Chenshi Ji);**

**卢峭林(化学方向, Qiaolin Lu);**

**林可(生物方向,Ke Lin)**

**Tutor: 徐昊 AT**

**Professor Hao Xu, Professor Adriano Tavares**

**Contents of Project…………………………………….2**

1. **Some information about this project…………………………………………………………3**
2. **Problem Statement……………………………………………………………………………4**
3. **Analysis………………………………………………………………………………………..4**
4. **Design…………………………………………………………………………………….……4**
5. **Implementation……………………………………………………………………………….4**

**5.1 Basic Functions……………………………………………………………………………5**

**5.2 Playlist …………..…………………………………………………………………………9**

**5.3 Lyrics………………………………………………………………………………………14**

**5.4 UI Design…………………………………………………………………………………...19**

**6. Test……………………………………………………………………………………………...32**

1. **Some information about this project:**
   1. **Project name: gds Music Player (“高大上”音乐播放器)**
   2. **Difficulty Level: 94-96**
   3. **Developmental language and developmental tools: C Language, windows operating system, Microsoft Visual Studio 2015, Microsoft DirectX SDK.**
   4. **Group members and their own respective work:** 
      1. **Qiaolin Lu (chemistry in TAQ): basic functions about playing, pausing, stopping;**
      2. **Chenshi Ji (biology in TAQ): the whole UI designing and other things;**
      3. **Ke Lin (biology in TAQ): functions about lyric and playlist;**
   5. **What can this player do?**

**Be able to**

**1.5.1 play most of songs;**

**1.5.2 play songs by m3u song list;**

**1.5.3 play mp3 songs;**

**1.5.4 play, pause, stop;**

**1.5.5 play next and last song;**

**1.5.6 display the title of the song;**

**1.5.7 play next song automatically when the player has finished the last song;**

**1.5.8 display the lyric when the player is playing the song;**

**1.5.9 show the time of the process;**

**1.5.10 add one song and the song list;**

**1.5.11 delete and save the song of playlist;**

**2. Problem Statement**

**Our problem is to make a music player using C language.**

**From the user's point of view, it should have basic functions of a music player. The user will be able to select a file from his computer and play it. He can know how long the song is and how much time has passed since it began. The song can be paused in the middle and start from where it stopped.**

**From developers’ point of view, he must enable the player to import files stored in memory, before which the APP needs to filter file format first. To make it more user friendly, a playlist should be added to help the user choose the songs imported. The list is able to be saved for the user. We can further optimize the program by adding lyric which can change automatically according to the playtime. Lastly, we should create user interface containing pushbuttons, box for the list and texts of lyric. More detail problems are allowed to be added during program making.**

1. **Analysis**

**3.1 To make a successful music player application, we should achieve following functions to give the user best experience:**

**3.1.1 Open the folder to add songs and lyric;**

**3.1.2 Make a list and arrangement it;**

**3.1.3 User can control the playing program like pausing, stopping, playing, deleting, next playing and so on.**

**For this APP, we use follow basic functions to get it:**

**3.2.1 Use API and Windows function to link to the button and control the songs;**

**3.2.2 Create Linked list, Structure and Array to make Lyric and Item;**

**3.2.3 Use handle to control the process.**

1. **Design**

**According to Problem Statement and Analysis, we have separated our work into three parts, basic functions about playing, pausing, stopping, playlist and lyrics, and UI design.**

1. **Implementation** 
   1. **Basic Functions**
      1. **System Framework and Flow Chart**

gdsPlayFile

PT\_CTROL

CheckThread

InitPlay

mpPlay1

DestroyThread

PlayThread

* + 1. **At first, we define gdsPlayFile function to create a thread to control the function of playing, pausing and some basic functions. The main functions will be achieved through PlayThread() this function. The whole process will end when threadid as a global variable works.**

**gdsPlayFile:**

1. void gdsPlayFile(void \*path)
2. {
3. if (h)
4. {
5. PostThreadMessage(threadid, PT\_QUIT, (WPARAM)0, (LPARAM)0);
6. WaitForSingleObject(h, INFINITE);
7. }
8. h = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)PlayThread, path, 0, (LPDWORD)&threadid);
9. if (NULL == h)
10. {
11. printf("create thread error\n");
12. return;
13. }
14. }
15. void InitPlay()
16. {
17. HRESULT hr = 0;
18. hr = CoInitialize(NULL);
19. if (FAILED(hr))
20. {
21. printf("init com error\n");
22. return;
23. }
24. hr = CoCreateInstance(&CLSID\_FilterGraph, NULL, CLSCTX\_INPROC\_SERVER, &IID\_IGraphBuilder, (void \*\*)&pGraph);
25. if (FAILED(hr))
26. {
27. printf("create fgm error\n");
28. return ;
29. }
30. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaControl, (void \*\*)&pControl);
31. if (FAILED(hr))
32. {
33. printf("get control error\n");
34. return ;
35. }
36. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaEvent, (void \*\*)&pEvent);
37. if (FAILED(hr))
38. {
39. printf("get event error\n");
40. return;
41. }
42. hr =IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaSeeking, (void \*\*)&pSeek);
43. if (FAILED(hr))
44. {
45. printf("get seek error\n");
46. return;
47. }
48. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IBasicAudio, (void \*\*)&pBA);
49. if (FAILED(hr))
50. {
51. printf("get basic audio error\n");
52. return;
53. }
54. }
    * 1. **Using InitPlay to initialize the play file**🡪 **then the player carries out the mpPlay1**🡪 **waiting to control while in the message loop body.**
55. void InitPlay()
56. {
57. HRESULT hr = 0;
58. hr = CoInitialize(NULL);
59. if (FAILED(hr))
60. {
61. printf("init com error\n");
62. return;
63. }
64. hr = CoCreateInstance(&CLSID\_FilterGraph, NULL, CLSCTX\_INPROC\_SERVER, &IID\_IGraphBuilder, (void \*\*)&pGraph);
65. if (FAILED(hr))
66. {
67. printf("create fgm error\n");
68. return ;
69. }
70. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaControl, (void \*\*)&pControl);
71. if (FAILED(hr))
72. {
73. printf("get control error\n");
74. return ;
75. }
76. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaEvent, (void \*\*)&pEvent);
77. if (FAILED(hr))
78. {
79. printf("get event error\n");
80. return;
81. }
82. hr =IGraphBuilder\_QueryInterface(pGraph, &IID\_IMediaSeeking, (void \*\*)&pSeek);
83. if (FAILED(hr))
84. {
85. printf("get seek error\n");
86. return;
87. }
88. hr = IGraphBuilder\_QueryInterface(pGraph, &IID\_IBasicAudio, (void \*\*)&pBA);
89. if (FAILED(hr))
90. {
91. printf("get basic audio error\n");
92. return;
93. }
94. }
    * 1. **gdsPlay: play**

**gdsPause: pause**

**gdsStop: stop**

* + 1. **send the instruction command of playing, pausing, stopping to the thread of control**

**5.1.6 Get the current position and final position by IMediaSeeking\_GetPositions, and compare the number. If they are equivalent, it means the processing of playing has ended and it will send PT\_QUIT command to thread and this song’s play will end.**

1. void CheckThread(void \*param)
2. {
3. HRESULT hr;
4. while (1)
5. {
6. long long stoppos = 0;
7. long long curpos = 0;
8. Sleep(1000);
9. if (pSeek == NULL)
10. {
11. return;
12. }
13. hr = IMediaSeeking\_GetPositions(pSeek, &curpos, &stoppos);
14. if (curpos == stoppos)
15. {
16. PostThreadMessage(threadid, PT\_QUIT, (WPARAM)0, (LPARAM)0);
17. return;
18. }
19. }
20. }
    * 1. **DestroyPlay function: free the object**
21. void DestroyPlay()
22. {
23. IBasicAudio\_Release(pBA);
24. pBA = NULL;
25. IMediaSeeking\_Release(pSeek);
26. pSeek = NULL;
27. IMediaControl\_Release(pControl);
28. pControl = NULL;
29. IMediaEvent\_Release(pEvent);
30. pEvent = NULL;
31. IGraphBuilder\_Release(pGraph);
32. pGraph = NULL;
33. CoUninitialize();
    1. **Playlist**

**5.2.1 Create an array and linelist to store the song by the time of addition and store in the memory allocation .**

struct item

{

char Item[256];

item \*next;

};

item \*g\_itemhead = NULL;

item \*g\_itemtail = NULL;

int number = 0;

int g\_itemnumber = 0;

char g\_wrongitem[256] = {0};

int CreateItem()

{

FILE \*fp;

char wrongitem[256] = {0};

char \*way;

char buf[256] = {0};

GetModuleFileName(NULL, wrongitem, 256);

way = strrchr(wrongitem, '\\');

way[1] = '\0';

strcpy(g\_wrongitem, wrongitem);

sprintf(wrongitem, "%s%s", wrongitem, Wrongitem);

fp = fopen(wrongitem, "r");

if (fp == NULL)

{

printf("Couldn't open\n",Wrongitem);

return -1;

}

do

{

char \*p;

item \*line;

p = fgets(buf, 256, fp);

if (p == NULL)

{

break;

}

g\_itemnumber++;

line = malloc(sizeof(item));

memset(line->Item, 0, MAX\_PATH);

strncpy(line->Item, buf, strlen(buf) - 1); //not read \n

line->next = NULL;

if (g\_itemhead == NULL)

{

g\_itemhead = line;

g\_itemtail = g\_itemhead;

}

else

{

g\_itemtail->next =line;

g\_itemtail = line;

}

} while (1);

return 0;

}

**5.2.2 Release the Whole Line list and delete single song.**

int DestroyWrongItem()

{

item \*line;

item \*linenext;

if (g\_itemhead == NULL)

{

return 0;

}

line = g\_itemhead;

while (line)

{

linenext = line->next;

free(line);

line = linenext;

}

return 0;

}

int DeleteItem(int pos)

{

item \*line;

item \*lineprev;

int i = 0;

if (pos >= g\_itemnumber || g\_itemhead == NULL)

{

return 0;

}

number++;

// only one in the list

if (g\_itemnumber == 1)

{

free(g\_itemhead);

g\_itemhead = NULL;

g\_itemtail = NULL;

g\_itemnumber=g\_itemnumber-1;

return 0;

}

// delete the first one

if (pos == 0)

{

line = g\_itemhead;

g\_itemhead = line->next;

free(line);

line = NULL;

return 0;

}

// delete the last one

if (pos == g\_itemnumber -1)

{

line = g\_itemhead;

while (line->next->next)

{

line = line->next;

}

g\_itemtail = line;

line = line->next;

g\_itemtail->next = NULL;

free(line);

line = NULL;

g\_itemnumber=g\_itemnumber-1;

return 0;

}

// delete middle one

lineprev = g\_itemhead;

while (i < pos - 1)

{

lineprev =lineprev->next;

i++;

}

line = lineprev->next;

free(line);

line = NULL;

lineprev->next = NULL;

g\_itemnumber--;

return 0;

}

**5.2.3 Analyze the path and address of the playing song**

int AddItem(char \*itempath)

{

WIN32\_FIND\_DATA finddata;

HANDLE handle;

handle = FindFirstFile(itempath, &finddata);

if (INVALID\_HANDLE\_VALUE == handle)

{

return -1;

}

if (finddata.cFileName)

{

item \*line;

line = malloc(sizeof(item));

memset(line->Item, 0, 256);

strcpy(line->Item, itempath);

line->next = NULL;

if (g\_itemhead == NULL)

{

g\_itemhead = line;

g\_itemtail = g\_itemhead;

}

else

{

g\_itemtail->next = line;

g\_itemtail = line;

}

g\_itemnumber++;

number++;

}

return 0;

}

**5.2.4 Save the playing song to the folder of the app**

int Save()

{

FILE \*fp;

item \*line;

char List[256] = {0};

if (number == 0 || g\_itemnumber == 0 || g\_itemhead == NULL)

{

return 0;

}

sprintf(List, "%s%s", g\_wrongitem, Wrongitem);

fp = fopen(List, "w+");

if (fp == NULL)

{

printf("open %s error\n", Wrongitem);

return -1;

}

line = g\_itemhead;

while (line)

{

fputs(line->Item, fp);

fputs("\n", fp);

line = line->next;

}

fflush(fp);

fclose(fp);

return 0;

}

* 1. **Lyrics**
     1. **First we make a structure to store the message including lyric , time and length of it. Then make an linelist to achieve show the lyric one by one**

1. struct Libret
2. {
3. int startTime;
4. char Lib[256];
5. Libret \*next;
6. };
7. Libret \*g\_Head = NULL;
8. Libret \*g\_Tail = NULL;
9. int g\_Count = 0;
   * 1. **Get the lyric into the linelist and analyze the information. At the end of it, arrange it by the time 。**
10. fp = fopen(way, "r");
11. if (fp == NULL)
12. {
13. printf("lyric is error(%d)\n", way, GetLastError());
14. return -1;
15. }
16. while ((p = fgets(buf, 256, fp)) != NULL)
17. {
18. char \*tok = "[";
19. char \*store;
20. if (\*p == '\n')
21. {
22. continue;
23. }
24. store = strchr(p, '[');
25. store++;
26. if (strnname(store, "ar", 2) == 0)
27. {
28. }
29. else if (strnname(store, "ti", 2) == 0)
30. {
31. }
32. else if (strnname(store, "al", 2) == 0)
33. {
34. }
35. else if (strnname(store, "by", 2) == 0)
36. {
37. }
38. else if (strnname(store, "offset", 6) == 0)
39. {
40. }
41. else
42. {
43. char \*lrc;
44. Libret \*llnode = NULL;
45. float mi= 0.0;
46. float ss = 0.0;
47. int Time= 0;
48. char \*l = p;
49. char \*r;
50. lrc = strrchr(buf, ']');
51. lrc++;
52. do
53. {
54. char member[128] = {0};
55. l = strL(l, '[');
56. if (l == NULL)
57. {
58. break;
59. }
60. l++;
61. r = strL(left, ':');
62. memcpy(member, l, r - l);
63. mi = atof(member);
64. l = r+1;
65. r = strL(l, ']');
66. memcpy(member, l, r - l);
67. ss = atof(member);
68. Time = (mi\*60+ss);
70. llnode = malloc(sizeof(Libret));
71. memset(llnode, 0, sizeof(Libret));
72. llnode->startTime = Time;
73. strcpy(llnode->Lib, lrc);
74. llnode->next = NULL;
75. if (g\_Head == NULL)
76. {
77. g\_Head = llnode;
78. g\_Tail = llnode;
79. }
80. else
81. {
82. Libret \*llprev = g\_Head;
83. Libret \*llnext = llprev->next;
84. if (llnode->startTime < llprev->startTime)
85. {
86. llnode->next = llprev;
87. g\_Head = llnode;
88. }
89. else
90. {
91. while (llnext && llnext->startTime <= llnode->startTime)
92. {
93. llprev = llnext;
94. llnext = llnext->next;
95. }
96. llnode->next = llnext;
97. llprev->next = llnode;
98. if (llnext == NULL)
99. {
100. g\_Tail = llnode;
101. }
102. }
103. }
104. g\_Count++;
105. left = right+1;
106. } while (1);
107. }
108. }
109. return 0;
110. }

**5.3.3 Get the Lyric(Libret) by time and position**

1. char \*GetLibretByTime(int startTime)
2. {
3. Libret \*llnode;
4. char \*szLrc = NULL;
5. if (g\_Head == NULL)
6. {
7. return NULL;
8. }
9. llnode = g\_Head;
10. while (llnode)
11. {
12. if (llnode->startTime == startTime)
13. {
14. if (szLrc == NULL)
15. {
16. szLrc = malloc(strlen(llnode->Lib)+1);
17. memset(szLrc, 0, strlen(llnode->Lib)+1);
18. strcat(szLrc, llnode->Lib);
19. }
20. else
21. {
22. szLrc = realloc(szLrc, strlen(szLrc)+strlen(llnode->Lib)+2);
23. strcat(szLrc, " ");
24. strcat(szLrc, llnode->Lib);
25. }
26. }
27. llnode = llnode->next;
28. }
29. return szLrc;
30. }
31. char \*GetLibretByPos(int pos)
32. {
33. Libret \*llnode;
34. int i = 0;
35. if (pos > g\_Count || g\_Head == NULL)
36. {
37. return NULL;
38. }
39. llnode = g\_Head;
40. while (llnode)
41. {
42. if (i == pos)
43. {
44. break;
45. }
46. llnode = llnode->next;
47. i++;
48. }
49. if (llnode == NULL)
50. {
51. return NULL;
52. }
53. return llnode->Lib;
54. }
    * 1. **Relase the linelist for next song**
55. int Release()
56. {
57. Libret \*llnode;
58. if (g\_Head == NULL)
59. {
60. return 0;
61. }
62. llnode = g\_Head;
63. while (llnode)
64. {
65. Libret \*llnext = llnode->next;
66. free(llnode);
67. llnode = llnext;
68. }
69. g\_Head = NULL;
70. g\_Tail = NULL;
71. g\_Count = 0;
72. return 0;
73. }

**LOWORD(wParam)**

**5.4 UI Design**

**WinMain**

**(APIENTRY)**

**GetWindowRect**

**createBtn**

**createLyricStatic**

**createListBox**

**createTimeStatic**

**WM\_PAINT**

**……**

**WNDCLASSEX wcex**

**WM\_CREATE**

**WndProc**

**message loop**

**WM\_COMMAND**

**Call functions according to the ID**

**LOWORD(wParam)**

**5.4.1 Create window: here we start from the entry point, WinMain and register a window class.**

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine, //命令行command line

int nCmdShow) //display mode

{

MSG msg;//信息

HWND hWnd;//画布

char szTitle[] = "GDSPLAYER"; // The title bar text

WNDCLASSEX wcex = { 0 };

g\_hInst = hInstance;

wcex.cbSize = sizeof(WNDCLASSEX); //size of WNDCLASSEX

wcex.style = CS\_HREDRAW | CS\_VREDRAW; //redraw when position changes

wcex.lpfnWndProc = (WNDPROC)WndProc; //message-handler function, to call CALLBACK WndProc（LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)）

wcex.hInstance = 0; //handle to the current instance

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1); //background color

wcex.lpszClassName = "gdsclass"; //参窗口类名Pprameter window class name

wcex.hIcon = LoadIcon(hInstance, (LPCTSTR)105); //icon

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW); //cursor

wcex.lpszMenuName = 0; //MenuName

wcex.hIconSm = 0;

RegisterClassEx(&wcex); //regester

hWnd = CreateWindowEx(WS\_EX\_ACCEPTFILES, "gdsclass", szTitle, WS\_OVERLAPPED | WS\_CAPTION | WS\_SYSMENU | WS\_MINIMIZEBOX | WS\_CLIPCHILDREN | WS\_VISIBLE, //创建窗口

CW\_USEDEFAULT, CW\_USEDEFAULT, whWi, whHe, NULL, NULL, 0, NULL);

if (!hWnd)

{

return FALSE;

}

g\_hWnd = hWnd;

while (GetMessage(&msg, NULL, 0, 0)) // message loop, windows keep showing

{

TranslateMessage(&msg); //converts virtual keys to character messages

DispatchMessage(&msg); //the dispatch message calls the callback function分派消息调用回调函数

}

return msg.wParam;

}

**5.4.2 Create buttons\list\static**

void createBtn(HWND hWnd, LPARAM lParam)

{

int xPos, yPos;

xPos = 0;

yPos = 425;

btnPrev = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "prev", WS\_VISIBLE | WS\_CHILD, xPos + 5, yPos + 0, 45, 30, hWnd, C\_BTN\_PREV, g\_hInst, NULL);

btnPlay = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "play", WS\_VISIBLE | WS\_CHILD, xPos + 55, yPos + 0, 45, 30, hWnd, C\_BTN\_PLAY, g\_hInst, NULL);

btnNext = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "next", WS\_VISIBLE | WS\_CHILD, xPos + 105, yPos + 0, 45, 30, hWnd, C\_BTN\_NEXT, g\_hInst, NULL);

btnStop = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "stop", WS\_VISIBLE | WS\_CHILD, xPos + 155, yPos + 0, 45, 30, hWnd, C\_BTN\_STOP, g\_hInst, NULL);

xPos = whWi + 20;

btnAdd = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "add", WS\_VISIBLE | WS\_CHILD, xPos - 235, yPos, 45, 30, hWnd, C\_BTN\_ADD, g\_hInst, NULL);

btnDir = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "dir", WS\_VISIBLE | WS\_CHILD, xPos - 185, yPos, 45, 30, hWnd, C\_BTN\_DIR, g\_hInst, NULL);

btnDel = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "del", WS\_VISIBLE | WS\_CHILD, xPos - 135, yPos, 45, 30, hWnd, C\_BTN\_DEL, g\_hInst, NULL);

btnSave = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_BUTTON, "save", WS\_VISIBLE | WS\_CHILD, xPos - 85, yPos, 45, 30, hWnd, C\_BTN\_SAVE, g\_hInst, NULL);

}

void createListBox(HWND hWnd)

{

int i = 0;

POINT p = {10, 70};

int ident = 0;

int plItemCount = 0;

int colWidth = 0;

int colMaxWidth = 300;

RECT rect;

GetWindowRect(hWnd, &rect);

PlayListInit();//Initializes the default playlist

plItemCount = GetDefaultPlaylistTotalItem();

playlistLB = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_LISTBOX, "default",

WS\_VISIBLE | LBS\_COMBOBOX | WS\_CHILD|WS\_VSCROLL |LBN\_DBLCLK|LBS\_NOTIFY,

1, 90, whWi-18, rect.bottom - rect.top - 170, hWnd, C\_BTN\_FROM\_PL, g\_hInst, NULL);//create

for (i = 0; i < plItemCount; i++)

{

char \*val = NULL;

char \*pval = NULL;

char plItem[MAX\_PATH] = {0};

val = GetItemFromDefaultPlaylist(i);//Gets all the song paths and inserts them into the playlist component获取所有歌曲路径并将其插入到播放列表组件中

pval = strrchr(val, '\\');

if (pval == NULL)

{

pval = val;

}

else

{

pval++;

}

sprintf(plItem, "%03d.%s", i+1, pval);

SendMessage(playlistLB, LB\_INSERTSTRING, (-1), plItem);

}

}

void createLyricStatic(HWND hWnd)

{

RECT rect;

GetWindowRect(hWnd, &rect);

lyricStatic = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_STATIC, "lyric",

WS\_VISIBLE | WS\_CHILD | LBS\_NOTIFY | SS\_SUNKEN | SS\_CENTER, 1, 35, whWi - 18, rect.bottom - rect.top - 450, hWnd, C\_LYRIC\_STATIC, g\_hInst, NULL);

}

void createTimeStatic(HWND hWnd)

{

timeStatic = CreateWindowEx(WS\_EX\_ACCEPTFILES, WC\_STATIC, "TIME: \n00:00 / 00:00",

WS\_VISIBLE | WS\_CHILD|LBS\_NOTIFY|SS\_CENTER, whWi-150, 0, 100, 30, hWnd, C\_TIME\_STATIC, g\_hInst, NULL);

}

**5.4.3 Corresponding functions**

int addFile(HWND hWnd)

{

OPENFILENAME fn;

char filefilter[] =

"All Supported files\0\*.mp1;\*.mp2;\*.mp3;\*.m3u;\*.ogg;\*.pls;\*.wav\0MPEG audio files (\*.mp1;\*.mp2;\*.mp3)\0\*.mp1;\*.mp2;\*.mp3\0Vorbis files (\*.ogg)\0Playlist files (\*.m3u;\*.pls)\0\*.m3u;\*.pls\0WAV files (\*.wav)\0\*.wav\0All Files (\*.\*)\0\*.\*\0";

BOOL retVal = FALSE;

char initialfilename[MAX\_PATH \* 100] = "";

fn.lStructSize = sizeof(OPENFILENAME);

fn.hwndOwner = hWnd;

fn.hInstance = NULL;

fn.lpstrFilter = filefilter;

fn.lpstrCustomFilter = NULL;

fn.nMaxCustFilter = 0;

fn.nFilterIndex = 0;

fn.lpstrFile = initialfilename;

fn.nMaxFile = MAX\_PATH \* 200;

fn.lpstrFileTitle = NULL;

fn.nMaxFileTitle = 0;

fn.lpstrInitialDir = "./";

fn.lpstrTitle = NULL;

fn.Flags =

OFN\_ALLOWMULTISELECT | OFN\_HIDEREADONLY | OFN\_EXPLORER |

OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST | OFN\_ENABLESIZING;

fn.nFileOffset = 0;

fn.nFileExtension = 0;

fn.lpstrDefExt = NULL;

fn.lCustData = 0;

fn.lpfnHook = NULL;

fn.lpTemplateName = NULL;

retVal = GetOpenFileName(&fn);//Get the added file path

if (retVal != FALSE)

{

char path\_buffer[MAX\_PATH];

char \*pval = NULL;

strcpy(path\_buffer, fn.lpstrFile);

pval = strrchr(path\_buffer, '\\');

if (pval == NULL)

{

pval = path\_buffer;

}

else

{

pval++;

}

SendMessage(playlistLB, LB\_INSERTSTRING, (-1), pval);//Send the song name to the playlist playlistBL component

DefaultPlaylistAddItem(path\_buffer);//Add the song path to the default playlist

}

return 0;

}

int addDir(HWND hWnd)

{

BROWSEINFO browseinfo;

LPITEMIDLIST itemlist;

int image = 0;

char directorychoice[MAX\_PATH];

char fullpath[MAX\_PATH];

HANDLE found;

WIN32\_FIND\_DATA finddata;

char pathbuf2[MAX\_PATH];

char dirBuf[MAX\_PATH];

browseinfo.hwndOwner = hWnd;

browseinfo.pidlRoot = NULL;

browseinfo.pszDisplayName = directorychoice;

browseinfo.lpszTitle = "Choose a directory to add";

browseinfo.ulFlags = BIF\_EDITBOX;

browseinfo.lpfn = NULL;

browseinfo.lParam = 0;

browseinfo.iImage = image;

itemlist = SHBrowseForFolder(&browseinfo);

if (itemlist == NULL)

{

return 1;

}

SHGetPathFromIDList(itemlist, dirBuf);

if (dirBuf[strlen(dirBuf) - 1] == '\\'

&& strcmp(dirBuf, "\\") != 0) dirBuf[strlen(dirBuf) - 1] =

'\0';

strcpy(fullpath, dirBuf);

if (strcmp(fullpath, "\\") == 0)

strcat(fullpath, ".\\\*.mp3");

else

strcat(fullpath, "\\\*.mp3");

found = FindFirstFile(fullpath, &finddata);

do {

char somepath[MAX\_PATH];

strcpy(somepath, dirBuf);

if (strcmp(somepath, "\\") == 0)

strcpy(somepath, "\\.");

sprintf(pathbuf2, "%s\\%s", somepath, finddata.cFileName);

if ((finddata.cFileName[0] != '.'

&& finddata.cFileName[1] != 0)

&& (finddata.cFileName[0] != '.'

&& finddata.cFileName[1] != '.'

&& finddata.cFileName[2] != 0))

{

char \*pval = NULL;

pval = strrchr(pathbuf2, '\\');

if (pval == NULL)

{

pval = pathbuf2;

}

else

{

pval++;

}

SendMessage(playlistLB, LB\_INSERTSTRING, (-1), pval);

DefaultPlaylistAddItem(pathbuf2);

}

} while (FindNextFile(found, &finddata));

FindClose(found);

return 0;

}

int delFile(HWND hWnd)

{

int lbItem = -1;

lbItem = (int)SendMessage(playlistLB, LB\_GETCURSEL, 0, 0);//Gets the index of the currently selected song

if (lbItem != -1)

{

DefaultPlaylistDeleteItem(lbItem); // Deletes songs from the playlist and items from the playlist component

SendMessage(playlistLB, LB\_DELETESTRING, lbItem, 0);

}

return 0;

}

void timeProc(UINT wTimerID, UINT msg,DWORD dwUser,DWORD dwl,DWORD dw2)

{

long long curpos = 0;

long long endpos = 0;

int curtime = 0;

int endtime = 0;

char \*p = NULL;

char \*szLryic = NULL;

char szTime[1024] = {0};

double rate = 0.0;

gdsGetPositions(&curpos, &endpos);

if (curpos == endpos)

{

// play the next

WCHAR songpath[MAX\_PATH] = {0};

char lyricpath[MAX\_PATH] = {0};

char \*p;

playIndex++;

if (playIndex > GetDefaultPlaylistTotalItem())

{

playIndex = 0;

}

p = GetItemFromDefaultPlaylist(playIndex);

if (p == NULL)

{

return;

}

getLyricPath(p, lyricpath);

LyricDestroy();

LyricInit(lyricpath);

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, p, strlen(p), songpath, MAX\_PATH);

gdsPlayFile(songpath);

status = C\_BTN\_PLAY;

SetWindowText(btnPlay, "pause");

SetWindowText(g\_hWnd, p);

}

curtime = (int)(curpos / (double)10000000);

endtime = (int)(endpos / (double)10000000);

sprintf(szTime, "TIME: \n%02d:%02d / %02d:%02d", curtime/60, curtime%60, endtime/60, endtime%60);

SendMessage(timeStatic, WM\_SETTEXT, 0, szTime);

szLryic = GetLyricByStartTime(curtime);

if (szLryic != NULL)

{

SendMessage(lyricStatic, WM\_SETTEXT, 0, szLryic);

}

}

**5.4.4 The corresponding handler for this form, where we get the component ID via LOWORD(wParam) and realize function**

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

{

PAINTSTRUCT ps;

HDC hdc;

RECT rect;

static HDC s\_hdcMem;

static RECT oldRect;

switch (message)

{

case WM\_PAINT: //Redraw message

{

hdc = BeginPaint(hWnd, &ps);

GetWindowRect(hWnd, &rect);

if (rect.right - rect.left != oldRect.right - oldRect.left

|| rect.bottom - rect.top != oldRect.bottom - oldRect.top)

{

oldRect = rect;

SetWindowPos(playlistLB, NULL, 1, 90, whWi - 18, oldRect.bottom - oldRect.top - 170, SWP\_SHOWWINDOW);

SetWindowPos(lyricStatic, NULL, 1, 35, whWi - 18, oldRect.bottom - oldRect.top - 450, SWP\_SHOWWINDOW);

}

EndPaint(hWnd, &ps);

}

break;

case WM\_CREATE:

{

GetWindowRect(hWnd, &oldRect);

createBtn(hWnd, lParam);

createLyricStatic(hWnd);

createListBox(hWnd);

createTimeStatic(hWnd);

}

break;

case WM\_COMMAND:

{

int id = LOWORD(wParam);//get buttons' ID

switch (id)

{

case C\_BTN\_PREV:

{

char \*p = NULL;

static WCHAR songpath[MAX\_PATH] = { 0 };

char lyricpath[MAX\_PATH] = { 0 };

memset(songpath, 0, MAX\_PATH);

playIndex--;

if (playIndex < 0)

{

playIndex = GetDefaultPlaylistTotalItem();

}

p = GetItemFromDefaultPlaylist(playIndex);

if (p == NULL)

{

break;

}

getLyricPath(p, lyricpath);

LyricDestroy();

LyricInit(lyricpath);

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, p, strlen(p), songpath, MAX\_PATH);

endTime();

gdsPlayFile(songpath);

beginTime();

status = C\_BTN\_PLAY;

SetWindowText(btnPlay, "pause");

SetWindowText(hWnd, p);

}

break;

case C\_BTN\_PLAY:

{

if (status != C\_BTN\_PLAY )

{

gdsPlay();

status = C\_BTN\_PLAY;

SetWindowText(btnPlay, "pause");

}

else if (status == C\_BTN\_PLAY)

{

gdsPause();

status = C\_BTN\_PAUSE;

SetWindowText(btnPlay, "play");

}

}

break;

case C\_BTN\_STOP:

{

if (status == C\_BTN\_PLAY || status == C\_BTN\_PAUSE)

{

gdsStop();

status = C\_BTN\_STOP;

SetWindowText(btnPlay, "play");

}

}

break;

case C\_BTN\_NEXT:

{

char \*p = NULL;

static WCHAR songpath[MAX\_PATH] = {0};

char lyricpath[MAX\_PATH] = {0};

memset(songpath, 0, MAX\_PATH);

playIndex++;

if (playIndex > GetDefaultPlaylistTotalItem())

{

playIndex = 0;

}

p = GetItemFromDefaultPlaylist(playIndex);

if (p == NULL)

{

break;

}

getLyricPath(p, lyricpath);

LyricDestroy();

LyricInit(lyricpath);

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, p, strlen(p), songpath, MAX\_PATH);

endTime();

gdsPlayFile(songpath);

beginTime();

status = C\_BTN\_PLAY;

SetWindowText(btnPlay, "pause");

SetWindowText(hWnd, p);

}

break;

case C\_BTN\_FROM\_PL:

{

int selid = HIWORD(wParam);//message type

switch (selid)

{

case LBN\_DBLCLK://double click

{

// Get selected index.

int lbItem = (int)SendMessage(playlistLB, LB\_GETCURSEL, 0, 0); //Gets the double-clicked entry index

char \*p = NULL;

static WCHAR songpath[MAX\_PATH] = {0};

char lyricpath[MAX\_PATH] = {0};

memset(songpath, 0, MAX\_PATH);

playIndex = lbItem;

p = GetItemFromDefaultPlaylist(playIndex);//Gets the path of the song to play

if (p == NULL)

{

break;

}

getLyricPath(p, lyricpath); //Convert the song name to the lyrics name

LyricDestroy();

LyricInit(lyricpath);// initialize the lyrics

MultiByteToWideChar(CP\_ACP, MB\_PRECOMPOSED, p, strlen(p), songpath, MAX\_PATH);

endTime();

gdsPlayFile(songpath); //Begin playing the song

beginTime(); // create a timer beginTime() to check the play progress

status = C\_BTN\_PLAY;//Set the play button state

SetWindowText(btnPlay, "pause");

SetWindowText(hWnd, p);

}

break;

default:

{

}

break;

}

}

break;

case C\_BTN\_ADD:

{

addFile(hWnd);

}

break;

case C\_BTN\_DIR:

{

addDir(hWnd);

}

break;

case C\_BTN\_DEL:

{

delFile(hWnd);

}

break;

case C\_BTN\_SAVE:

{

DefaultPlaylistSave();

}

break;

default:

break;

}

}

break;

case WM\_DESTROY: //WINDOWS destroy massage

PostQuitMessage(0);

default:

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

}

return 0;

}

1. **Test**



Timer

Lyrics

Playlist

Scroll Bar

Songs

Buttons

**This is our UI based on our code. As you can see, if we can click relevant button, corresponding function will work. For example, if we click “dir” button, a list of songs will be added, likewise, the song will be played while clicking the “play” button. If the songs and lyrics are in the same directory, we can also easily find the lyrics when it plays.**

|  |  |
| --- | --- |
| **Components** | **Functions** |
| **Prev button** | **Play the last music** |
| **Play** | **play** |
| **Next** | **Play the next music** |
| **Stop** | **Stop playing** |
| **Add** | **Add a music** |
| **Dir** | **Add a playlist** |
| **Del** | **Delete a music** |
| **Save** | **Save** |
| **Timer** | **Show time of playing** |
| **Lyrics** | **Show the lyrics** |
| **Playlist** | **Show the playlist** |
| **Scroll Bar** | **Help to show the playlist** |