```
#include "Timer.h"

void CALLBACK BringTimerUp(void *hWnd, BOOLEAN timerOrWaitFired)
{
    PostMessage((HWND)hWnd, WM_SETTIME, TMR_ADDSEC, NULL);
}

// Set the timer queue for bringing the timer up
DWORD _stdcall TimerUp(void *hWnd)
{
    HANDLE timer = NULL;
    HANDLE queue = CreateTimerQueue();
    CreateTimerQueueTimer(&timer, queue, &BringTimerUp, hWnd, 999, 1000, 0);
    PINFO info = (PINFO)GetWindowLongPtr((HWND)hWnd, GWLP_USERDATA);
    info->hTimerQueue = queue;
    return 0;
}

void CALLBACK BringTimerDown(void *hWnd, BOOLEAN timerOrWaitFired)
{
    PostMessage((HWND)hWnd, WM_SETTIME, TMR_SUBSEC, NULL);
}

// Set the timer queue for bring the timer down
DWORD _stdcall TimerDown(void *hWnd)
{
    HANDLE timer = NULL;
    HANDLE queue = CreateTimerQueue();
    CreateTimerQueueTimer(&timer, queue, &BringTimerDown, hWnd, 999, 1000, 0);
    PINFO info = (PINFO)GetWindowLongPtr((HWND)hWnd, GWLP_USERDATA);
    info->hTimerQueue = queue;
    return 0;
}

// Beep!
DWORD _stdcall Beep(void *null)
{
    Beep(988, 300);
    return 0;
}

BOOL Beep(void)
{
    HANDLE beeper = CreateThread(NULL, 16, &Beep, NULL, 0, NULL);
    CloseHandle(beeper);
    return beeper ? TRUE : FALSE;
}

char GetNumChar(BYTE num)
{
    return (num >= 1 && num <= 9) ? num + 48 : '0';
}

// Stop the timer.
void StopTimer(PINFO info)
{
    if (info->hTimerQueue) {
        DeleteTimerQueueEx(info->hTimerQueue, INVALID_HANDLE_VALUE);
        info->hTimerQueue = NULL;
    }
    if (info->hTimer) {
        TerminateThread(info->hTimer, 1);
        CloseHandle(info->hTimer);
        info->hTimer = NULL;
    }
}
```

```
67
68 // Handles windows messages.
69 LRESULT _stdcall WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
70 {
71     // Define variables
72     LRESULT lResult = 0;
73     PAINTSTRUCT ps;
74     PINFO info;
75     PTIME time;
76     char *lpcTime;
77     switch (msg)
78     {
79     case WM_SETTIME:
80     case WM_LBUTTONDOWN:
81     case WM_RBUTTONDOWN:
82     case WM_RESET:
83     case WM_PAINT:
84     case WM_CLOSE:
85     case WM_CHAR:
86         // All of these messages need this info
87         info = (PINFO)GetWindowLongPtr(hWnd, GWLP_USERDATA);
88     }
89     switch (msg)
90     {
91     case WM_SETTIME:
92         // Sets the time on the timer.
93         // wParam says whether to add or subtract a second.
94         if (wParam == TMR_ADDSEC) {
95             // Make sure the timer is set to ON
96             info->isSet = TRUE;
97             // Set the new time
98             time = info->pTime;
99             if (time->seconds + 1 < 60)
100                time->seconds++;
101            else {
102                time->seconds = 0;
103                if (time->minutes + 1 < 60)
104                    time->minutes++;
105                else {
106                    time->minutes = 0;
107                    if (time->hours + 1 < 100)
108                        time->hours++;
109                    else SendMessage(hWnd, WM_SETTIME, TMR_RESET, NULL);
110                }
111            }
112            // If the timer is done, beep
113            if (time->seconds == 0 && time->minutes == 0 && time->hours == 0)
114                Beep();
115        }
116        else if (wParam == TMR_SUBSEC) {
117            // Make sure the timer is ON
118            info->isSet = TRUE;
119            // Set the new time
120            time = info->pTime;
121            if (time->seconds > 0)
122                time->seconds--;
123            else {
124                time->seconds = 59;
125                if (time->minutes > 0)
126                    time->minutes--;
127                else {
128                    time->minutes = 59;
129                    if (time->hours > 0)
130                        time->hours--;
131                }
132            }
```

```
133             // If the timer is done, beep, set the timer to OFF, and reset the timer
134             if (time->seconds == 0 && time->minutes == 0 && time->hours == 0) {
135                 Beep();
136                 info->isOn = FALSE;
137                 SendMessage(hWnd, WM_RESET, NULL, NULL);
138             }
139         }
140         else {
141             // If wParam is not a valid value,
142             // leave the function and report an error (with lResult)
143             if (wParam != TMR_RESET && wParam != TMR_SET) {
144                 lResult = 2;
145                 break;
146             }
147             // Releas the memory from the old time, it's no longer needed
148             if (info->pTime)
149                 free(info->pTime);
150             // If the time is being set to a specific
151             // time, get that time from lParam
152             if (wParam == TMR_SET) {
153                 time = (PTIME)lParam;
154                 if (info->pBaseTime)
155                     free(info->pBaseTime);
156                 info->pBaseTime = time;
157             }
158             else time = info->pBaseTime;
159             if (!time) {
160                 time = NEW(TIME);
161                 if (!time) {
162                     lResult = 1;
163                     break;
164                 }
165                 if (info->pBaseTime)
166                     free(info->pBaseTime);
167                 info->pBaseTime = time;
168             }
169             time = NEW(TIME);
170             if (!time) {
171                 lResult = 1;
172                 break;
173             }
174             time->hours = info->pBaseTime->hours;
175             time->minutes = info->pBaseTime->minutes;
176             time->seconds = info->pBaseTime->seconds;
177             // check if it should go down or up, and set it to OFF
178             info->isGoingDown = time->hours != 0 || time->minutes != 0 || time->seconds != 0;
179             info->isSet = FALSE;
180         }
181         // Take the time, and convert it into a string
182         info->pTime = time;
183         lpcTime = new char[9];
184         if (!lpcTime) {
185             lResult = 1;
186             break;
187         }
188         if (info->lpcTime)
189             delete [] info->lpcTime;
190         lpcTime[0] = GetNumChar(time->hours / 10);
191         lpcTime[1] = GetNumChar(time->hours % 10);
192         lpcTime[2] = ':';
193         lpcTime[3] = GetNumChar((time->minutes / 10) % 6);
194         lpcTime[4] = GetNumChar(time->minutes % 10);
195         lpcTime[5] = ':';
196         lpcTime[6] = GetNumChar((time->seconds / 10) % 6);
197         lpcTime[7] = GetNumChar(time->seconds % 10);
198         lpcTime[8] = '\0';
```

```
199         // info->lpcTime is what WM_PAINT uses to print the time
200         info->lpcTime = lpcTime;
201         InvalidateRect(hWnd, NULL, FALSE);
202         break;
203     case WM_PAINT:
204         TextOut(BeginPaint(hWnd, &ps), 5, 5, info->lpcTime, 8);
205         EndPaint(hWnd, &ps);
206         break;
207     case WM_LBUTTONDOWN:
208         // If the timer is on, stop it; otherwise, start it up
209         if (info->isOn) {
210             StopTimer(info);
211             info->isOn = FALSE;
212         }
213         else {
214             info->hTimer = CreateThread(NULL, 0, info->isGoingDown ? &TimerDown : &TimerUp, hWnd, 0,    ↙
    NULL);
215             info->isSet = TRUE;
216             info->isOn = TRUE;
217         }
218         break;
219     case WM_RBUTTONDOWN:
220         // If the timer is not is it's reset state, reset it
221         if (!info->isSet)
222             break;
223     case WM_RESET:
224         // Stop the timer and reset it; if it was going, keep it going
225         StopTimer(info);
226         if (info->isOn) {
227             info->hTimer = CreateThread(NULL, 0, info->isGoingDown ? &TimerDown : &TimerUp, hWnd, 0,    ↙
    NULL);
228         }
229         SendMessage(hWnd, WM_SETTIME, TMR_RESET, NULL);
230         break;
231     case WM_CHAR:
232         // Get a char typed by the user; see if it's a number.
233         // If it is, set the next timer value to it.
234         if (wParam < '0' || wParam > '9' || (wParam > '5' && (info->wTypedTimePlace == 2 ||
235             info->wTypedTimePlace == 4)))
236             break;
237         wParam -= '0';
238         time = NEW(TIME);
239         if (!time) {
240             lResult = 1;
241             break;
242         }
243         time->hours = info->pTime->hours;
244         time->minutes = info->pTime->minutes;
245         time->seconds = info->pTime->seconds;
246         // Change the time based on the next place
247         switch (info->wTypedTimePlace)
248         {
249         case 0:
250             time->hours = time->hours % 10;
251             time->hours += wParam * 10;
252             break;
253         case 1:
254             time->hours -= time->hours % 10;
255             time->hours += wParam;
256             break;
257         case 2:
258             time->minutes = time->minutes % 10;
259             time->minutes += wParam * 10;
260             break;
261         case 3:
262             time->minutes -= time->minutes % 10;
```

```
263             time->minutes += wParam;
264             break;
265         case 4:
266             time->seconds = time->seconds % 10;
267             time->seconds += wParam * 10;
268             break;
269         case 5:
270             time->seconds -= time->seconds % 10;
271             time->seconds += wParam;
272         }
273         if (info->wTypedTimePlace > 4)
274             info->wTypedTimePlace = 0;
275         else info->wTypedTimePlace++;
276         SendMessage(hWnd, WM_SETTIME, TMR_SET, (LPARAM)time);
277         SendMessage(hWnd, WM_RESET, NULL, NULL);
278         break;
279     case WM_CLOSE:
280         // Clean up
281         if (info->pTime)
282             free(info->pTime);
283         if (info->pBaseTime)
284             free(info->pBaseTime);
285         StopTimer(info);
286         free(info);
287         PostQuitMessage(0);
288         break;
289     default:
290         return DefWindowProc(hWnd, msg, wParam, lParam);
291     }
292     if (lResult == 1)
293         MessageBox(hWnd, "There was an error allocating memory.", "Timer", MB_ICONERROR);
294     return lResult;
295 }
296
297 // Checks if code is an argument in the given command line.
298 // Syntax: "Timer.exe -[arg]"
299 BOOL IsArg(char *lpCmdLine, char *code)
300 {
301     UINT nCode = 0;
302     while (code[nCode])
303         nCode++;
304     if (nCode > 0)
305         for (char c; c = *lpCmdLine++; )
306             if (c == '-' && *lpCmdLine == *code) {
307                 WORD nArg = 0;
308                 while (lpCmdLine[nArg] && lpCmdLine[nArg] != ' ')
309                     nArg++;
310                 if (nCode != nArg)
311                     continue;
312                 BOOL eq = TRUE;
313                 char *codeCpy = code;
314                 char *lpCmdLineCpy = lpCmdLine;
315                 for (char c; c = *++codeCpy; )
316                     if (c != *++lpCmdLineCpy) {
317                         eq = FALSE;
318                         break;
319                     }
320                 if (eq)
321                     return TRUE;
322             }
323     return FALSE;
324 }
325
326 // Since this program is based off a GUI, it uses
327 // WinMain instead of int main for the Windows OS.
328 int _stdcall WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, char *lpCmdLine, int nCmdShow)
```

```
329 {
330     // Create the window
331     HBRUSH hBg = CreateSolidBrush(RGB(255, 255, 255));
332     HCURSOR hCur = LoadCursor(NULL, IDC_HAND);
333     HICON hIconSm = (HICON)LoadImage(hInst, "IDI_ICON", IMAGE_ICON, 16, 16, LR_SHARED);
334     HICON hIcon = LoadIcon(hInst, "IDI_ICON");
335     if (!hBg || !hCur || !hIconSm || !hIcon) {
336         MessageBox(NULL, "There was an error creating an object.", "Timer", MB_ICONERROR);
337         return 1;
338     }
339     WNDCLASSEX wc;
340     wc.cbSize = sizeof(WNDCLASSEX);
341     wc.cbClsExtra = 0;
342     wc.cbWndExtra = 0;
343     wc.hbrBackground = hBg;
344     wc.hCursor = hCur;
345     wc.lpfnWndProc = WndProc;
346     wc.hIcon = hIcon;
347     wc.hIconSm = hIconSm;
348     wc.hInstance = hInst;
349     wc.lpszClassName = "Timer";
350     wc.lpszMenuName = NULL;
351     wc.style = NULL;
352     if (!RegisterClassEx(&wc)) {
353         MessageBox(NULL, "There was an error registering the class.", "Timer", MB_ICONERROR);
354         return 2;
355     }
356     HWND hWnd = CreateWindow("Timer", "Timer", WS_SYSMENU | WS_MINIMIZEBOX, CW_USEDEFAULT, 0,
357         165, 55, NULL, NULL, hInst, NULL);
358     if (!hWnd) {
359         MessageBox(NULL, "There was an error creating the window.", "Timer", MB_ICONERROR);
360         return 3;
361     }
362     // Create the thread that changes the time, but don't start it yet
363     HANDLE thread = CreateThread(NULL, 0, &TimerUp, (HWND)hWnd, CREATE_SUSPENDED, NULL);
364     if (!thread) {
365         MessageBox(NULL, "There was an error creating the timer.", "Timer", MB_ICONERROR);
366         return 4;
367     }
368     // Set up the class holding all the info about the timer
369     PINFO info = NEW(INFO);
370     PTIME time = NEW(TIME);
371     if (!info || !time) {
372         MessageBox(NULL, "There was an error allocating memory.", "Timer", MB_ICONERROR);
373         return 5;
374     }
375     info->pBaseTime = time;
376     info->hTimer = thread;
377     // So that this data can be accessed as long as the HWND of the GUI is
378     // known, set the timer info as the user data for the window.
379     SetWindowLongPtr((HWND)hWnd, GWLP_USERDATA, (LONG)info);
380     // Check for arguments for fading.
381     // -f = fade both in and out
382     // -fi = fade in
383     // -fo = fade out
384     // Note: this feature was designed on Windows 7 starter,
385     // which doesn't fade windows in or out.
386     BOOL bFade = IsArg(lpCmdLine, "f");
387     if ((bFade || IsArg(lpCmdLine, "fi")) && AnimateWindow(hWnd, 200, AW_BLEND | AW_ACTIVATE))
388         SendMessage(hWnd, WM_SETTIME, TMR_RESET, NULL);
389     else {
390         SendMessage(hWnd, WM_SETTIME, TMR_RESET, NULL);
391         ShowWindow(hWnd, nCmdShow);
392     }
393     // Set up the message loop.
394     MSG msg;
```

```
395     while (GetMessage(&msg, NULL, 0, 0)) {
396         TranslateMessage(&msg);
397         DispatchMessage(&msg);
398     }
399     if (!bFade && !IsArg(lpCmdLine, "fo") || !AnimateWindow(hWnd, 200, AW_BLEND | AW_HIDE))
400         ShowWindow(hWnd, SW_HIDE);
401     // Clean up
402     DeleteObject(hBg);
403     DestroyCursor(hCur);
404     DestroyIcon(hIcon);
405     DestroyIcon(hIconSm);
406     return msg.wParam;
407 }
```