Writing a Macro Recorder/Player using Win32 Journal Hooks

S. Senthil Kumar, 19 Jun 2004

Rate:

4.85 (50 votes)

This article informs you how to use Win32 Journal hooks to write a simple Macro Recorder/Player.

Download source files - 22.6 Kb

Download executable - 54.1 Kb

Introduction

This article is about using journal hooks to write a simple Macro Recorder/Player. It's kinda different from the macro stuff seen in MS Office© products in that it works across applications. So you can record a macro, that opens a text file in Notepad, copies the address of a URL, opens IE, and navigates to the URL. Once recorded, you can play it back any number of times. All this stuff is possible because of the magic of journal hooks, which is a facility provided by Windows to record and playback keyboard and mouse events.

General Win32 Hooking

Win32 hooking is a way of monitoring/filtering messages passed to applications. Hooks may be systemwide or thread specific. Systemwide hooks can monitor/filter messages passed to any application. There are several types of hooks like keyboard hooks, mouse motion hooks, CBT hooks etc. All types of hooks are set using the `SetWindowsHookEx` function, which essentially takes a callback function and the type of hook as the parameter. Callback functions for systemwide hooks must reside in a DLL, the only exceptions are journal hooks.

Hide Copy Code

```
HHOOK SetWindowsHookEx(int idHook, HOOKPROC lpfn,
                       HINSTANCE hMod, DWORD dwThreadId);
```

`idHook` is the type of hook, `lpfn` is the address of the callback function, `hMod` is the handle of the DLL (if any) containing the callback function, and `dwThreadID` if zero, is used to indicate system wide hooking. Once this is done, every time a particular message/event relevant to the hook happens, Windows gets back to your function (`lpfn`) and you can do the necessary processing. More than one application could have set a systemwide hook, in that case, Windows maintains a chain of hooks with the most recently installed hook at the head of the chain. Because your code doesn't know whether any more hooks are down the line, it's always better to call `CallNextHookEx` with whatever parameters were passed to your callback function. The signature of all callback functions, irrespective of the type of hook, is:

Hide Copy Code

```
LRESULT CALLBACK FilterFunc( int nCode, WORD wParam, DWORD lParam );
```

Of course, how `wParam` and `lParam` are interpreted varies according to the type of hook.

If installed as a systemwide hook, the hook remains there till the application containing the hook procedure closes, or Windows shuts down. If you want to unhook by yourself, you need to use `UnhookWindowsHookEx`, passing it the `HHOOK` value returned by `SetWindowsHookEx`.

Hide Copy Code

```
BOOL UnhookWindowsHookEx( HHOOK hhk);
```

Journal Hooks

Journal hooks are special systemwide hooks provided by Windows, specifically to record and playback keyboard and mouse events. Because it is systemwide, it doesn't matter to which application a particular keystroke was directed at, your callback will still get it. To use journal hooks, you need to call `SetWindowsHookEx` twice, once for recording (`idHook` = `WH_JOURNALRECORD`) and once for playback (`idHook` = `WH_JOURNALPLAYBACK`).

Let's see how we can use these to write a Macro recorder/player. The general idea is this:

To record,

- Set a `WH_JOURNALRECORD` hook.
- Ask the user to do whatever he wants to record, and in the hook procedure, save the user actions.
- Unhook the `WH_JOURNALRECORD` hook and write the saved actions to a file.

To playback,

- Set the `WH_JOURNALPLAYBACK` hook.
- Whenever a callback function call comes, read an entry from the saved file and assign it to an out parameter of the callback function.
- Unhook after playback is complete.

Recording

Let's dive into the code now. This is the implementation of the hook procedure for recording a macro:

Hide Shrink Copy Code

```
LRESULT CALLBACK JournalRecordProc(int code,WPARAM wparam,LPARAM lparam)
{
    //The recording callback hook function

    SHORT val=GetKeyState(VK_PAUSE);

    //If Pause/Break key is pressed, stop recording
    //The other key sequence to stop recording,
    //Ctrl+Esc automatically evicts the hook
    //so all we have to do is update the UI
    //accordingly, that code is there in
    //the GetMsgProc function
    if (val & 0x80000000)
    {
        dialog_ptr->StopRecording();
        return CallNextHookEx(hook_handle,code,wparam,lparam);
    }

    EVENTMSG *mesg=NULL;

    switch (code)
    {
```

```
    case HC_ACTION:
        //The main part. HC_ACTION is called whenever
        //there is a mouse or keyboard event.
        //lparam is a pointer to EVENTMSG struct,
        //which contains the necessary information
        //to replay the message.

        mesg=(EVENTMSG *)lparam;

        //Subtract message tick count from
        //the start of recording tick count
        //Subtracted value gives the relative delay
        //between start of recording and
        //current message. We use this at playback
        //to reproduce this delay.
        mesg->time=mesg->time-start_time;

        //Write out the EVENTMSG struct to file
        //(output is a fstream pointer)
        output->write((const char *)mesg,sizeof(EVENTMSG));
        output->flush();

        break;
    default:
        return CallNextHookEx(hook_handle,code,wparam,lparam);
    }
    return 0;
}
```

For the moment, ignore the first few lines, the real action starts from the `switch` block. Every time a keyboard or mouse event occurs, Windows calls `JournalRecordProc` with code as `HC_ACTION`. When code is `HC_ACTION`, `lParam` is a pointer to a `EVENTMSG` structure, with the following fields:

Hide Copy Code

```
typedef struct {
    UINT message;
    UINT paramL;
    UINT paramH;
    DWORD time;
    HWND hwnd;
} EVENTMSG, *PEVENTMSG;
```

`message` is used to differentiate between keyboard and mouse event messages. The meaning of `paramL` and `paramH` vary depending on the message. For example, for mouse events, it will hold the X and Y coordinates of the current mouse location. `time` gives the system tick count (the value of a call to `GetTickCount()`) when the event occurred, and `hwnd` is the window handle to which the message was directed at.

For our purposes, we needn't worry about the contents of the structure, we just write out the structure (in binary mode) to a file. The only processing that I've done is changing the `time` field, and we'll see why a bit later. And just as I had mentioned before, I've been a good hooking citizen and called `CallNextHookEx` if the code is something other than `HC_ACTION`.

Now, for the strange code at the top. You have to somehow allow the user to inform you that he's finished with recording. The code at the top allows the user to stop recording by getting the current state of Pause/Break key. If it's pressed (the first bit of the return value is one), then I do the cleanup operation, i.e., unhooking and closing the file.

Playback

The hook procedure for playback is slightly more complex. Here's the code for the hook procedure for playback (`WH_JOURNALPLAYBACK`):

```
LRESULT CALLBACK JournalPlaybackProc(int code,WPARAM wparam,LPARAM lparam)
{
    EVENTMSG *mesg;

    static EVENTMSG mesg_from_file;

    LRESULT delta;

    switch(code)
    {
    case HC_GETNEXT:


        //Like JournalRecordProc, this function also
        //has the lparam, except that we must
        //supply the EVENTMSG structure. The system uses the
        //info in the EVENTMSG to replay the message.

        mesg=(EVENTMSG *)lparam;

        //HC_GETNEXT can be called as many times as
        //the system wants, we have to supply the
        //same message. Only when HC_SKIP is called,
        //we can advance to the next message in our file.
        //move_next is the BOOL variable that takes care of this.
        if (move_next)
        {
            if (next_message_exists)
            {
                input->read((char *)&mesg_from_file,sizeof(EVENTMSG));
                //If this is EOF, reset next_message_exists
                if (input->eof()) next_message_exists=FALSE;

            }
            else
            {
                //Re initialize start_time to current tick count so
                //that messages can be played back
                //with the same delays between them.
                start_time=GetTickCount();

                //Reset everything else
                mesg_count=0;
                move_next=TRUE;
                next_message_exists=TRUE;

                return 0;
                        }

            //This is so that the same message is sent,
            //until HC_SKIP is called
            move_next=FALSE;
        }

        //Copy mesg from file to the lparam
        //structure (which has been typecast)
        mesg->hwnd=mesg_from_file.hwnd ;
        mesg->message=mesg_from_file.message ;
        mesg->paramH=mesg_from_file.paramH ;
        mesg->paramL=mesg_from_file.paramL ;

        mesg->time= start_time + mesg_from_file.time;
```

```
        //This measures the difference between
        //when mesg is to played and current time.
        delta=mesg->time-GetTickCount();

        //If more than zero (ie mesg is to played some time in the future)
        // return that time. The system sleeps for that time, before
        //calling HC_GETNEXT again, in which case,
        //delta will be <=0, and we return 0.
        //The system then plays that message.
        if (delta>0)
        {

            return delta;
        }
        else
        {
            return 0;
        }

        break;

    case HC_SKIP:
        //Move to the next message
        move_next=TRUE;
        mesg_count++;

        break;
    default:
            return CallNextHookEx(hook_handle,code,wparam,lparam);
    }
    //System ignores this. Just to satisfy the compiler.
    return 0;
}
```

Once you do SetWindowsHookEx with idHook as WH_JOURNALPLAYBACKPROC, Windows takes control of the keyboard and the mouse. It then calls your hook procedure with code as HC_GETNEXT or HC_SKIP. While you read the information in the EVENTMSG structure during recording, you fill the EVENTMSG structure (by casting the lParam parameter) during playback. Windows then uses that information to determine what event needs to be sent.

A minor detail here is that Windows can call your hook procedure with code as HC_GETNEXT any number of times, you're supposed to supply the same EVENTMSG structure. Only when a call with code as HC_SKIP comes, you're allowed to supply a different structure (the next event to be played). The code above does that using the move_next BOOL variable, which is set to TRUE inside HC_SKIP. Inside HC_GETNEXT, the code reads the next EVENTMSG structure only if move_next is TRUE, otherwise it returns the same structure. It isn't that you must supply only information previously recorded using WH_JOURNALRECORDPROC, as long as the EVENTMSG structure is filled with valid data. This opens up other ways to use journal hooks, for instance, you can use the playback journal hook to send keyboard and mouse events to a particular window.. but that's stuff for another article.

Coming back to our macro recorder, if you had noticed carefully, I would have copied all the fields in the structure read from the file to the EVENTMSG structure, except the time field. While recording also, the time field would have been modified before being written into the file. Here's the reason why.

During playback, Windows gets to know the time interval between successive events based on the value returned from the playback hook procedure when called with code as HC_GETNEXT. If it's non-zero, it sleeps for that many milliseconds before calling your hook procedure with HC_GETNEXT again. It does this repeatedly, until you return a zero. After playing that event, it comes back to your hook procedure with HC_SKIP and so on. Assume that we're going to playback events at exactly the same rate as it happened while recording. For this to happen, I record the relative time at which each event occurred, in the time field while recording. So that's why, I had to do:

Hide Copy Code

```
mesg->time=mesg->time-start_time;
```

`start_time` is the value of `GetTickCount()` when recording started, and `mesg->time` is the tick count value when the event occurred, so the difference gives me the value as to when the event occurred calculated from the start of recording.

I can now use this information for playback. I record the value of `GetTickCount()` when playback started, and I use that information to determine when the event is to be played back.

Hide Copy Code

```
mesg->time= start_time + mesg_from_file.time;
```

We need to calculate how much time to sleep before this message is to be processed. For that, I calculate the difference between the calculated `mesg->time` value and the current tick count. If it's less than zero, that means we're actually lagging behind, no worry, we just return zero so that the event gets to be played immediately. If the difference is non-zero, I return that value, so Windows sleeps for that much time. When it then gets back to the hook function, the difference this time will be surely lesser than the previous one (actually, should be zero, but sometimes not), as `GetTickCount()` would give a higher value. At one point, it will get to zero or less than zero, and that's when the event gets played.

Hide Copy Code

```
//This measures the difference between
//when mesg is to played and current time.
delta=mesg->time-GetTickCount();

//If more than zero (ie mesg is to played some time in the future)
// return that time. The system sleeps for that time, before
//calling HC_GETNEXT again, in which case,
//delta will be <=0, and we return 0.
//The system then plays that message.
if (delta>0)
{
    return delta;
}
else
{
    return 0;
}
```

Other considerations

Because Windows takes control of the keyboard and mouse queues when the `WH_JOURNALPLAYBACK` hook is installed, a malicious application can use it to wreak havoc on your system. Pressing Ctrl+Alt+Del or Ctrl+Esc evicts the journal hook, thus stopping the recording/playback process midway. Our macro recorder/player would need to know about it, so that the UI reflects the fact that recording/playback has been stopped. Thankfully, Windows sends a `WH_CANCELJOURNAL` Windows message when it evicts the hook. The trouble is, the message doesn't have a Windows handle, i.e., it is not directed at any window, so how do we get to know about it? By now, the answer should be obvious, use another systemwide hook, this time a Windows Message Hook (`idHook` = `WH_GETMESSAGE`). This hook will get called whenever **any** Windows application calls `GetMessage` or `PeekMessage`, so we can catch `WH_CANCELJOURNAL` there and inform our UI that recording/playback has been interrupted. Here's the code for the `WH_GETMESSAGE` hook:

Hide Shrink Copy Code

```
LRESULT CALLBACK GetMsgProc(int code, WPARAM wparam, LPARAM lparam)
{
    //This hook is needed to respond to Ctrl+Break
    //or Ctrl+Esc or Ctrl+Alt+Del
    //which is used when the user wants to stop recording or playing back.
    //The system removes the hook anyway,
    //but WinMacro will not be aware of that
    //so we need to watch for all messages and
```

```
    //see if WM_CANCELJOURNAL is sent.
    //If so, we need to change our UI state
    //to indicate recording/playback has
    //been cancelled. I've also removed the hook,
    //but itsn't necessary, the system does it.
    if (code==HC_ACTION)
    {
        MSG *msg=(MSG *)lparam;
        if (msg->message==WM_CANCELJOURNAL)
        {
            //If user interrupted playback.
            if (playing_back)
            {
                uninstall_playback_hook();
                MessageBox(thiswindow_hwnd,
                  "Playback Interrupted","WinMacro v1.2",
                  MB_OK|MB_TASKMODAL|MB_ICONEXCLAMATION);
            }
            else if (recording)
            {
                //Indicate that recording must stop.
                dialog_ptr->StopRecording();
            }
        }
    }
    else
    {
        return CallNextHookEx(getmsg_hook_handle,code,wparam,lparam);
    }
return 0;
}
```

Something More!

That's all the stuff we need to know to write a simple macro recorder/player. In addition to whatever has been discussed, code in the download includes stuff like:

- Repeating Playback as many times as the user wants to - For this, I simply read the structures stored in the file into a list the first time playback occurs, after that, I supply records from the list.
- Slowing down or speeding up the playback. This requires tweaking the return value of the `JournalPlaybackProc`.

Using WinMacro

The UI of the macro recorder, which I've christened as WinMacro, is pretty simple. For recording, you type/browse to the name of the file you want to record to, and then click Record. For stopping the recording process, you hit Pause/Break. For playback, you type/browse to the name of a recorded file and hit Playback. The Options button opens a new dialog, that allows you to indicate the number of times to repeat playback, to set the playback speed, etc. For more information about how to use it, please visit WinMacro.

Limitations

For all that we've discussed till now, the macro recorder/player is pretty dumb. All it knows about are where you clicked the mouse and which key you pressed at a particular point in time, nothing more. Strange things will happen if the window coordinates differ between recording and playback. For instance, if you've recorded a macro, that opens Notepad, clicks on the File Menu, and then clicks Exit. Now, imagine what will happen if Notepad opens in a different location in the screen when you playback!! The mouse

clicks will go to whatever window is currently there in the recorded location.

The End

Anyway, I learnt a lot about Windows hooking while authoring this application. I initially wrote it just out of curiosity, but I figured it might be a pretty useful utility. I hope you'll agree . Happy Hooking!!

History

- Initial submission (19 Jun 2004).

License

Share

About the Author

S. Senthil Kumar

Software Developer Atmel R&D India Pvt. Ltd.
India

I'm a 27 yrs old developer working with Atmel R&D India Pvt. Ltd., Chennai. I'm currently working in C# and C++, but I've done some Java programming as well. I was a Microsoft MVP in Visual C# from 2007 to 2009.

You can read My Blog here. I've also done some open source software - please visit my website to know more.