

Format the Event Log

by Randy Charles Morin

Introduction

I answer a few thousands USENET post per year. It's something I take a lot of pride in. I really like knowing that I help others without any expectation of return. It's quite fulfilling. If you don't do it and have specialized knowledge, then you might consider doing the same.

Obviously, my answers resolve almost exclusively in areas where I have some expertise. The answer that I have the best responses tends to be with reading and formatting the Windows NT Event Log. In fact, my only solo book, "Programming Windows Services" devotes one of seven chapters to this one subject.

I guess you could say that my special skill is reading and formatting the Windows NT Event Log. Ok, that's not much of a skill. But I'll take my own pride in it.

As I wrote an entire chapter about this one skill, I have been discouraged from writing further on this subject. I mean, I wrote pretty much all there is to know. It's not really that much. But not all of you would have a convenient copy of my book on your desk and those of you who don't are missing those few pages that I wrote on this subject, that in themselves are the meaning of life. Ok, I admit, it's a few boring lines of code.

All that said this article is a rework of that chapter that explain in a few pages how you can call a few Win32 API functions and create your own Windows NT Event Log viewers.

Reading

Let's begin by trying to understand the ReadEventLog function.

```
BOOL ReadEventLog(  
    HANDLE hEventLog,           // handle to event log  
    DWORD dwReadFlags,         // how to read log  
    DWORD dwRecordOffset,      // initial record offset  
    LPVOID lpBuffer,           // buffer for read data  
    DWORD nNumberOfBytesToRead, // bytes to read  
    DWORD *pnBytesRead,        // number of bytes read  
    DWORD *pnMinNumberOfBytesNeeded // bytes required  
);
```

This function takes seven parameters. The first parameter is a handle to the event log that we want to read. This can be acquired by a simple call to the OpenEventLog function. The second parameter indicates how we want to read the event log. If you want to sequential read the event log back in time, from the latest record to the earliest record, then this parameter should be EVENTLOG_BACKWARDS_READ + EVENTLOG_SEQUENTIAL_READ. For sequential reading of the event log, the third parameter is ignored.

The fourth parameter is a buffer where the event log records should be written. The next parameter is the length in bytes of the buffer supplied as the fourth parameter. You have to specify the length in bytes, as each event log record varies in length.

The last two parameters are returned from the function and indicate the amount of bytes read and the amount of bytes required for the read. The last parameter is only important when the amount of bytes specified as the third parameter was not great enough to hold the entire next event log record.

When the ReadEventLog function returns, the buffer will point to one of more event log records. These records have the following format.

```
typedef struct _EVENTLOGRECORD {
    DWORD Length;
    DWORD Reserved;
    DWORD RecordNumber;
    DWORD TimeGenerated;
    DWORD TimeWritten;
    DWORD EventID;
    WORD EventType;
    WORD NumStrings;
    WORD EventCategory;
    WORD ReservedFlags;
    DWORD ClosingRecordNumber;
    DWORD StringOffset;
    DWORD UserSidLength;
    DWORD UserSidOffset;
    DWORD DataLength;
    DWORD DataOffset;
    //
    // Followed by:
    //
    // TCHAR SourceName[]
    // TCHAR Computername[]
    // SID UserSid
    // TCHAR Strings[]
    // BYTE Data[]
    // CHAR Pad[]
    // DWORD Length;
    //
} EVENTLOGRECORD, *PEVENTLOGRECORD;
```

What is very important is the Length member. This is important because you may read more than one record at a time. The subsequent records will start the Length of the previous record in bytes from the start of the previous record.

I created a class to encapsulate the EVENTLOGRECORD structure. Here, I automatically calculate the calculated offsets and return the concrete objects for each item in the EVENTLOGRECORD structure.

Listing 1: Event Log Record

```
#ifndef EventLogRecord_h
#define EventLogRecord_h

namespace kbcafe
{

class EventLogRecord
{
    const EVENTLOGRECORD * m_p;
    EventLogRecord()
        :m_p(NULL) { };
public:
    EventLogRecord(const EVENTLOGRECORD * p)
        :m_p(p) { };

    DWORD Length() { return m_p->Length; };

    DWORD RecordNumber() { return m_p->RecordNumber; };
};
};
```

```

DWORD TimeGenerated() { return m_p->TimeGenerated; };

DWORD TimeWritten() { return m_p->TimeWritten; };

DWORD EventID() { return m_p->EventID; };

WORD EventType() { return m_p->EventType; };

WORD EventCategory() { return m_p->EventCategory; };

std::string SourceName() { return (char*)m_p+56; };

std::string ComputerName()
    { return (char*)m_p+56+SourceName().length()+1; };

SID * UserSid() { return (SID*)m_p+m_p->UserSidOffset; };

DWORD UserSidLength() { return m_p->UserSidLength; };

std::vector<std::string> Strings()
{
    std::vector<std::string> v;
    DWORD offset = m_p->StringOffset;
    for (int i=0;i<m_p->NumStrings;i++)
    {
        std::string str = (char*)m_p+offset;
        v.push_back(str);
        offset += str.length()+1;
    }
    return v;
};

BYTE * Data() { return (BYTE*)m_p+m_p->DataOffset; };

DWORD DataLength() { return m_p->DataLength; };

};

};

#endif

```

Beyond this structure there is another issue with formatting the output of event log records. Each record is associated with a message string that can be found in a message module. These message strings provide additional information required in constructing the formatted output of an event log record.

Again, I created a class that encapsulates calls to various Win32 API functions (RegOpenKey, RegQueryValueEx, RegCloseKey, ExpandEnvironmentStrings, LoadLibraryEx, FormatMessage and FreeLibrary) that are required in constructing the format event log output. I won't explain the details of why you call these functions. You'll have to buy a copy of my book, for further details.

Listing 2: Format Record

```

#ifndef FormatEventLogRecord_h
#define FormatEventLogRecord_h
namespace kbcafe
{
class FormatEventLogRecord
{
    std::string m_eventlog;

    FormatEventLogRecord()
    { };
};

```

```

public:
    FormatEventLogRecord(const std::string & eventlog)
    {
        m_eventlog =
            "SYSTEM\\CurrentControlSet\\Services\\Eventlog\\"
            + eventlog;
    };

    std::string FormatRecord(const EVENTLOGRECORD * record)
    {
        std::stringstream ss;
        EventLogRecord rec(record);
        ss << m_eventlog << "\\\" << rec.SourceName();
        HKEY hkey;
        if (::RegOpenKey(HKEY_LOCAL_MACHINE, ss.str().c_str(),
            &hkey) != ERROR_SUCCESS)
        {
            return "";
        };
        char sz[256*256];
        DWORD dw = sizeof(sz);
        if (::RegQueryValueEx(hkey, "EventMessageFile", NULL,
            NULL, (BYTE*)sz, &dw) != ERROR_SUCCESS)
        {
            ::RegCloseKey(hkey);
            return "";
        }
        ::RegCloseKey(hkey);
        ::ExpandEnvironmentStrings(std::string(sz).c_str(),
            sz, sizeof(sz));
        HINSTANCE handle = ::LoadLibraryEx(sz, NULL, 0);
        if (handle == NULL)
        {
            return "";
        }
        DWORD dwArgs[16] = {0};
        DWORD offset = record->StringOffset;
        for (int i=0;i<record->NumStrings && i<16;i++)
        {
            dwArgs[i] = ((DWORD)record)+offset;
            std::string str = (TCHAR *)dwArgs[i];
            offset += str.length()+1;
        }
        dw = ::FormatMessage(FORMAT_MESSAGE_FROM_HMODULE |
            FORMAT_MESSAGE_ARGUMENT_ARRAY,
            handle, rec.EventID(), 0, sz, 1024,
            (va_list*)dwArgs);
        if (dw == 0)
        {
            dw = ::GetLastError();
        }
        ::FreeLibrary(handle);
        return sz;
    };
};

#endif

```

Finally with the two above classes, we can call the appropriate Win32 functions to complete the reading. First you have to call `OpenEventLog` to get a handle to a specific event log. The function takes two parameters. The first parameter is the name of the computer where the event log is located and the second is the name of the event log on that computer that you want to read. When you are finished with the handle to the event

log you should close it with the CloseEventLog Win32 API function. Finally you use the ReadEventLog Win32 API function to read records from the event log.

Listing 3: Read Event Log

```
#include <string>
#include <vector>
#include <sstream>
#include <iostream>
#define _WIN32_WINNT 0x0400
#include <windows.h>
#include "EventLogRecord.h"
#include "FormatEventLogRecord.h"

int main(int argc, char* argv[])
{
    std::string logname, computername;

    if (argc > 2)
    {
        logname = argv[1];
    }
    else if (argc == 3)
    {
        logname = argv[1];
        computername = argv[2];
    }
    else
    {
        std::cerr << "Usage:\n"
            "\tformateventlog logname [computername]";
        return 1;
    }

    HANDLE handle = ::OpenEventLog(computername.c_str(), logname.c_str());
    if (handle == NULL)
    {
        std::cerr << "OpenEventLog failed";
        return 1;
    };

    DWORD dwRecord = 0;
    static const int c = 65536;
    DWORD dwRead, dwNext;
    BYTE by[c];
    while (true)
    {
        if (!::ReadEventLog(handle, EVENTLOG_BACKWARDS_READ+
            EVENTLOG_SEQUENTIAL_READ, 1, by, c, &dwRead, &dwNext))
        {
            break;
        }

        DWORD dw = 0;
        while (dwRead > dw)
        {
            kbcafe::EventLogRecord rec((EVENTLOGRECORD *) (by+dw));
            std::cout << "Record Number = "
                << rec.RecordNumber() << "\n";
            std::cout << "Time Generated = "
                << rec.TimeGenerated() << "\n";
            std::cout << "Time Written = "
                << rec.TimeWritten() << "\n";
            std::cout << "Event ID = " << rec.EventID() << "\n";
            std::cout << "Event Type = "
                << rec.EventType() << "\n";
            std::cout << "Event Category = "
                << rec.EventCategory() << "\n";
            std::cout << "Source Name = "
```

```
        << rec.SourceName() << "\n";
std::cout << "Computer Name = "
        << rec.ComputerName() << "\n";

kbcafe::FormatEventLogRecord frec(logname);
std::cout << "Formatted = "
        << frec.FormatRecord((EVENTLOGRECORD *) (by+dw))
        << "\n" << std::endl;

    dw += rec.Length();
}
}
::CloseEventLog(handle);

return 0;
}
```

This code is missing one major feature. The Security log has a default message module. I haven't implemented this of yet. Maybe somebody can take a look and get back to me with a solution.

About the Author

Randy Charles Morin is the Lead Architect of SportMarkets Development from Toronto, Ontario, Canada and lives with his wife and two kids in Brampton, Ontario. He is the author of the www.kbcafe.com website, author of Wiley's Programming Windows Services book and co-author of many other programming books and articles.