Calling the Windows APIs for Large Files

CalvinH 18 Mar 2005 11:53 AM | 5

A customer was trying to use FoxPro to handle large files.

The files that Fox handles natively (like tables, indexes) are limited to using 32 bit addressing, (2^32 = 4 gigs). Back in the old days (not that long ago), there were no hard disks > 2 gigs. Who would ever need more?<g>

I've been using Virtual PC (a great product, which allows me to have many "virtual" computers exist on a single computer) and it creates files that are several gigabytes (they represent the entire hard disk of the virtual machine). I have a Win2003 Server virtual PC on my Windows XP machine and I can open multiple remote desktops to it!

Suppose I wanted to write a Fox program to read/write this large file. Not a problem: just call the 64 bit Win APIs (that were added when larger hard disks were invented).

For example, the SetFilePointer API is the 32 bit version, while SetFilePointerEx is the 64 bit version.

```
DWORD SetFilePointer(
  HANDLE hFile,
  LONG lDistanceToMove,
  PLONG lpDistanceToMoveHigh,
  DWORD dwMoveMethod
);


BOOL SetFilePointerEx(
  HANDLE hFile,
  LARGE_INTEGER liDistanceToMove,
  PLARGE_INTEGER lpNewFilePointer,
  DWORD dwMoveMethod
```

As you can see from the function signatures, the calls are almost identical: the only difference being LARGE_INTEGER in the place of LONG.

LONG is a 32 bit quantity, whereas LARGE_INTEGER is defined in Winnt.H as a Union

```
typedef union _LARGE_INTEGER {
    struct {
        DWORD LowPart;
        LONG HighPart;
    };
    struct {
        DWORD LowPart;
        LONG HighPart;
    } u;
    LONGLONG QuadPart;
} LARGE_INTEGER;

typedef LARGE_INTEGER *PLARGE_INTEGER;
```

The HighPart is 32 bit signed and the LowPart is 32 bit unsigned. Put them together and you have a 64 bit signed number.

When calling functions in the Win API, the parameters must be put on the process stack, which is 32 bits wide in 32 bit Windows XP.
The SetFilePointer call thus expects 4 32 bit words on the stack, because each parameter is of size 32 bits.

However, the SetFilePointerEx function expects 5 32 bit words on the stack: the LARGE_INTEGER is 2 32 bit words.
You may be thinking: what about the PLARGE_INTEGER ? isn't that 2 32 bit words too? Actually not: it's not a LARGE_INTEGER, but the address of a LARGE_INTEGER. Addresses in 32 bit Windows are 32 bits (surprise!).

So, how do we call such a function in Fox? We just pass 5 32 bit words on the stack by Declaring the LowPart and HighPart separately: See code below. Notice that the SetFilePointerEx Declare statement shows 5 parameters.

Fox just puts those parameters on the stack and the Win API call proceeds merrily along.

The sample code just opens a file and does a SEEK to a 64 bit offset into the file and shows the resulting position pointer.

(thanks to Jim Saunders for the problem and sample code).

```
#DEFINE CREATE_NEW                      1
#DEFINE CREATE_ALWAYS                   2
#DEFINE OPEN_EXISTING                   3
#DEFINE FILE_ATTRIBUTE_NORMAL         128
#DEFINE GENERIC_READ           2147483648  && 0x80000000
#DEFINE GENERIC_WRITE          1073741824  && 0x40000000
#DEFINE GENERIC_ALL             268435456  && 0x10000000
#DEFINE MAXIMUM_ALLOWED          33554432  && 0x02000000
#DEFINE STANDARD_RIGHTS_ALL       2031616  && 0x001F0000
#DEFINE FILE_SHARE_READ                 1
#DEFINE FILE_SHARE_WRITE                2
#DEFINE FILE_SHARE_DELETE               4
#DEFINE INVALID_HANDLE_VALUE           -1
#DEFINE  FILE_BEGIN  0
#DEFINE  FILE_CURRENT  1
#DEFINE  FILE_END  2
#DEFINE  MAXDWORD 4294967295

CLEAR
DECLARE INTEGER CreateFile IN kernel32 STRING    lpFileName,;
        INTEGER   dwDesiredAccess, INTEGER   dwShareMode,;
        INTEGER   lpSecurityAttr, INTEGER   dwCreationDisp,;
        INTEGER   dwFlagsAndAttrs, INTEGER   hTemplateFile

DECLARE INTEGER CloseHandle IN kernel32 INTEGER hObject
DECLARE long GetLastError in win32api

DECLARE long SetFilePointerEx IN kernel32;
    long hnd,;
    long lDistanceToMoveL,;
    long lDistanceToMoveH,;
    string @lpNewFilePointer,;
    long dwMoveMethod

filename = "E:\VirtualPCs\MyVirtualPCHardDisk.VHD"
fh = CreateFile(filename, ;
            GENERIC_READ,;
            FILE_SHARE_READ,;
            0,;
            OPEN_EXISTING,;
            FILE_ATTRIBUTE_NORMAL,;
            0)

 IF fh = INVALID_HANDLE_VALUE
  = MESSAGEBOX("Could not open "+filename,16,"")
  return
 ENDIF

cNewFilePointer = REPLICATE(CHR(0),8)

lDistanceToMove = 2^32+5
*lDistanceToMove = 2000
?"Dist=",lDistanceToMove
cpH=INT(lDistanceToMove/2^32)
cpL=lDistanceToMove - cpH * 2^32
?"High,Low=",cph,cpl
nret= SetFilePointerEx(fh, cpL,cpH, @cNewFilePointer, FILE_BEGIN)
*nret= SetFilePointerEx(fh, cpL,cpH, @cNewFilePointer, FILE_END)
IF nRet = 0
 ?"Error",GetLastError()
ENDIF
newfp=CToLI(cNewFilePointer)
?"Newfp=", newfp,LOG10(newfp)

=closehandle(fh)
CLEAR DLLS
RETURN

PROCEDURE CToLI(str as String) as Number  && str is an 8 byte string
```

```
      LOCAL num,i
      num=0
      FOR i = 0 TO 7
*           ?i,ASC(SUBSTR(str,i+1)),TRANSFORM(ASC(SUBSTR(str,i+1)),"@0x")
            num = num + 256^i * ASC(SUBSTR(str,i+1))
      ENDFOR
RETURN num
```

54461

Comments

余啊雷 28 Mar 2005 4:32 PM #

余啊雷 26 Sep 2006 2:13 AM #
I was browsing MSDN, and I came across this article: A Programmer's Perspective on NTFS 2000 Part 1:...

hellwood.ru 4 Sep 2008 5:38 AM #
<a href= http://hellwood.ru >Журнал о медицине</a>

Victor Savitskiy 19 Jan 2009 4:32 PM #
Calvin,

I've tried to write FGETS64() based on your code, but appeared that I might need FOPEN64(), FCLOSE64(), FREAD64(), FSEEK64() as well. Additionally, SetFilePointerEx() design triggers FGETS64() will be different from FGETS().

Some people are doing the same with Windows Scripting Host File System Object (WSH FSO). What is the benefit of using API instead of WSH FSO?

Frank 13 Jan 2011 1:51 AM #
How can I get for example -1 when using LowPart and HighPart?

I tried nret= SetFilePointerEx(fh, 1,-0, @cNewFilePointer, FILE_CURRENT), but it doesn't seem to work, because -0 is interpreted as 0.