If you want to use GUIDs to identify your files, then nobody's stopping you

devblogs.microsoft.com/oldnewthing/20110228-00

February 28, 2011

Raymond Chen

Igor Levicki proposes solving the problem of file extensions by using a GUID instead of a file name to identify a file.

You can do this already. Every file on an NTFS volume has an *object identifier* which is formally 16-byte buffer, but let's just call it a GUID. By default a file doesn't have an object identifier, but you can ask for one to be created with `FSCTL_CREATE_OR_GET_OBJECT_ID`, which will retrieve the existing object identifier associated with a file, or create one if there isn't one already. If you are a control freak, you can use `FSCTL_SET_OBJECT_ID` to specify the GUID you want to use as the object identifier. (The call fails if the file already has an object identifier.) And of course there is `FSCTL_GET_OBJECT_ID` to retrieve the object identifier, if any.

```
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <ole2.h>
#include <winioctl.h>
int __cdecl _tmain(int argc, PTSTR *argv)
{
 HANDLE h = CreateFile(argv[1], 0,
                FILE_SHARE_READ | FILE_SHARE_WRITE |
                FILE_SHARE_DELETE, NULL,
                OPEN_EXISTING, 0, NULL);
 if (h != INVALID_HANDLE_VALUE) {
  FILE_OBJECTID_BUFFER buf;
  DWORD cbOut;
  if (DeviceIoControl(h, FSCTL_CREATE_OR_GET_OBJECT_ID,
                NULL, 0, &buf, sizeof(buf),
                &cbOut, NULL)) {
    GUID guid;
    CopyMemory(&guid, &buf.ObjectId, sizeof(GUID));
    WCHAR szGuid[39];
    StringFromGUID2(guid, szGuid, 39);
    _tprintf(_T("GUID is %ws\n"), szGuid);
  }
  CloseHandle(h);
 }
 return 0;
}
```

This program takes a file or directory name as its sole parameter and prints the associated object identifier.

Big deal, now we have a GUID associated with each file.

The other half is, of course, using this GUID to open the file:

```
#define UNICODE
#define _UNICODE
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <ole2.h>
int __cdecl _tmain(int argc, PTSTR *argv)
{
 HANDLE hRoot = CreateFile(_T("C:\\"), 0,
                 FILE_SHARE_READ | FILE_SHARE_WRITE |
                 FILE_SHARE_DELETE, NULL,
                 OPEN_EXISTING,
                 FILE_FLAG_BACKUP_SEMANTICS, NULL);
 if (hRoot != INVALID_HANDLE_VALUE) {
  FILE_ID_DESCRIPTOR desc;
  desc.dwSize = sizeof(desc);
  desc.Type = ObjectIdType;
  if (SUCCEEDED(CLSIDFromString(argv[1], &desc.ObjectId))) {
   HANDLE h = OpenFileById(hRoot, &desc, GENERIC_READ,
                 FILE_SHARE_READ | FILE_SHARE_WRITE |
                 FILE_SHARE_DELETE, NULL, 0);
   if (h != INVALID_HANDLE_VALUE) {
    BYTE b;
    DWORD cb;
    if (ReadFile(h, &b, 1, &cb, NULL)) {
     _tprintf(_T("First byte of file is 0x%02x\n"), b);
    }
    CloseHandle(h);
   }
  }
  CloseHandle(hRoot);
 }
 return 0;
}
```

To open a file by its GUID, you first need to open something—anything—on the volume the file resides on. Doesn't matter what you open; the only reason for having this handle is so that `OpenFileById` knows which volume you're talking about. In our little test program, we use the C: drive, which means that the file search will take place on the C: drive.

Next, you fill in the `FILE_ID_DESCRIPTOR`, saying that you want to open the file by its object identifier, and then it's off to the races with `OpenFileById`. Just as a proof of concept, we read and print the first byte of the file that was opened as a result.

Notice that the file you open by its object identifier does not have to be in the current directory. It can be *anywhere on the C: drive*. As long as you have the GUID for a file, you can open it no matter where it is on the drive.

You can run these two programs just to enjoy the thrill of opening a file by its GUID. Notice that once you get the GUID for a file, you can move it anywhere on the drive, and `OpenFileById` will still open it.

(And if you want to get rid of those pesky drive letters, you can use the volume GUID instead. Now every file is identified by a pair of GUIDs: the volume GUID and the object identifier.)

So Igor's dream world where all files are referenced by GUID already exists. Why isn't everybody switching over to this utopia of GUID-based file identification?

You probably know the answer already: Because people prefer to name things with something mnemonic rather than a GUID. Imagine a file open dialog in this dream world. "Enter the GUID of the file you wish to open, or click Browse to see the GUIDs of all the files on this volume so you can pick from a list." How long would this dialog survive?

For today, you don't have to call me Raymond. You can call me {7ecf65a0-4b78-5f9b-e77c-8770091c0100}, or "91c" for short.

(And I've totally ignored the fact that using GUIDs to identify files does nothing to solve the problem of trying to figure out what program should be used to open a particular file.)

Bonus chatter: You can also open files by their file identifer, which is a volume-specific 64-bit value. But I chose to use the GUID both for the extra challenge, and just to show that Igor's dream world already exists.

Raymond Chen

Follow