How do I get from a file path to the volume that holds it?

devblogs.microsoft.com/oldnewthing/20201020-00

October 20, 2020

Raymond Chen

Say you have the path to a file and you want to access the volume that the file resides on.

Warning: All error checking is removed for expository purposes.

The first step on our journey is getting from the path to the volume mount point. This tells us where the root of the volume got inserted into the namespace.

```
TCHAR volumePath[MAX_PATH]; // for expository purposes
GetVolumePathName(filePath, volumePath, ARRAYSIZE(volumePath));
```

This information might be useful in its own right, but for us, it's just a stepping stone to the next piece of information: The volume name.

```
TCHAR volumeName[MAX_PATH]; // for expository purposes
GetVolumeNameForVolumeMountPoint(volumePath, volumeName, ARRAYSIZE(volumeName));
```

The volume name is returned in the form `\\?\Volume{guid}\`, with a trailing backslash. Note that this call will fail if the path is not a local drive.

Now things get weird.

If you pass that path to the `CreateFile` function with the trailing backslash intact, then you are opening a handle to the root directory of the volume.

If you pass that path to the `CreateFile` function with the trailing backslash removed, then you are opening a handle to the volume itself.

Depending on which operation you want to perform on the volume, you either must have or must not have that trailing backslash.

In our case, we want to open a handle to the volume itself, so we need to remove that trailing backslash. The call to `CreateFile` looks like this:

```
HANDLE handle = CreateFile(volumeNameWithoutTrailingBackslash,
    0, /* no special access requested */
    FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
    nullptr, /* no custom security */
    OPEN_EXISTING,
    FILE_FLAG_BACKUP_SEMANTICS,
    nullptr); /* template */
```

Volume query operations do not require any specific level of access, so we'll ask for no special access. Which is good, because regular non-elevated code doesn't really have much in the way of special access to volumes. You won't able to get `GENERIC_READ`, much less `GENERIC_WRITE`. (You'll be able to get `FILE_READ_ATTRIBUTES`, if that's any consolation.)

You also need to request backup semantics in order to open a volume.

We can put all of this together into a function called, say, `GetVolumeHandleForFile`. For RAII, I'm going to use wil.

```
wil::unique_hfile GetVolumeHandleForFile(PCWSTR filePath)
{
  wchar_t volumePath[MAX_PATH];
  THROW_IF_WIN32_BOOL_FALSE(GetVolumePathName(filePath,
                                volumePath, ARRAYSIZE(volumePath)));

  wchar_t volumeName[MAX_PATH];
  THROW_IF_WIN32_BOOL_FALSE(GetVolumeNameForVolumeMountPoint(volumePath,
                                volumeName, ARRAYSIZE(volumeName)));

  auto length = wcslen(volumeName);
  if (length && volumeName[length - 1] == L'\\')
  {
    volumeName[length - 1] = L'\0';
  }

  wil::unique_hfile result{ CreateFile(volumeName, 0,
                FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
                nullptr, OPEN_EXISTING, FILE_FLAG_BACKUP_SEMANTICS, nullptr) };
  THROW_LAST_ERROR_IF(!result);
  return result;
}
```

Now that you have the volume handle, you can ask the volume for information. We'll look at that next time.

Raymond Chen

Follow