Exiv2 - Feature #747

Direct FILE* access from FileIo interface

10 Dec 2010 11:44 - Adam Hooper

Status:	Closed	**Start date:**	10 Dec 2010
Priority:	Normal	**Due date:**	
Assignee:	Robin Mills	**% Done:**	100%
Category:	basicio	**Estimated time:**	0.00 hour
Target version:	0.26		

Description

I'm writing an object-oriented dcraw replacement and I want to handle metadata with Exiv2.

I don't want to use Exiv2 for File IO: it's too slow. I also don't want to open the same file twice (once to read metadata with Exiv2, once to read the RAW image data).

I think there should be a derivative of Exiv2::FileIo which would operate on an std::filebuf. That way, I'd be able to open an image file with my program, construct an Exiv2::FileBufIo() out of it, and when I'm done using Exiv2 to read its metadata, go on and use the std::filebuf for the rest of my work. (This implies the destructor wouldn't close the buffer.)

(Why std::filebuf and not FILE*? Well, because I'm selfish: that's what I'm using for file IO in the rest of my program.)

I suppose I could implement this in my own program, but I feel Exiv2 should have it.

Related issues:

Related to Exiv2 - Patch #1272: Possible issue with temp files being left beh...	**Closed**	**19 Jan 2017**

History

#1 - 19 Dec 2010 13:45 - Robin Mills

This sounds very reasonable to me. Have you thought about implementing this and donating the code to exiv2? I'm surprised by your claim about Exiv2 File IO being too slow. Can you elaborate?

#2 - 19 Dec 2010 15:20 - Adam Hooper

Robin Mills wrote:

> This sounds very reasonable to me. Have you thought about implementing this and donating the code to exiv2? I'm surprised by your claim about Exiv2 File IO being too slow. Can you elaborate?

1. Yes, I may implement and donate this.

2. Exiv2 file IO isn't too slow for reading metadata, but I assume it's too slow for reading RAW files. RAW decoders generally need to access a byte at a time, so the common case (i.e., "the next byte is already buffered") needs to be as quick as possible: branch, increment and fetch. Both std::filebuf's sgetc() and FILE*'s getc_unlocked() (_getc_nolock() on Windows) are usually inlined; Exiv2 uses a virtual method call. A buffer could be added on top of Exiv2's IO, but that would be inelegant.

#3 - 19 Dec 2010 17:14 - Andreas Huggel

Exiv2 is optimized to not read image data at all, not to read it fast. I just wonder whether there is a real gain in not closing the file and re-opening it, as opposed to using Exiv2's existing functions to read the metadata and your own for the image data. Suggest you do some prototyping to determine the performance difference and if it's worth your time before any change.

#4 - 19 Dec 2010 17:26 - Adam Hooper

This isn't about performance to me: it's about style.

I find it unwieldy to open the same file twice: that would make two different exceptions to catch in two different places. The second error would be unintuitive: if the file is deleted or the system runs out of file pointers while the program is running, it doesn't make sense that an error should be thrown after the file has already been opened successfully.

#5 - 19 Dec 2010 21:10 - Robin Mills

That's interesting. I'm working on a contract with one of the commercial licensee's of exiv2. I've been asked to replace their exif processing with exiv2. At the moment, they open a FILE* to read everything in the source image. Maybe I should consider creating a FileIO object using FILE* and submit that code to exiv2 (as required by the license).

So if Adam does std::filebuf, I'll do FILE* and everybody gets something. Is everybody happy with this proposal?

#6 - 19 Dec 2010 21:18 - Adam Hooper

Heh, works for me. Actually I'm considering dropping filebuf for my project in favour of FILE*: C (and Python and Vala and C# and ...) use FILE*, not filebuf.

(Incidentally, I've released a very-early version of my project. It's called "refinery" and it's at http://adamh.github.com/refinery. It has near-zero camera support right now and it's missing C bindings, but it has the fastest AHD interpolation I've seen.)

#7 - 19 Dec 2010 22:25 - Andreas Huggel

> Maybe I should consider creating a FileIO object using FILE*

Robin, what do you have in mind?

Exiv2::BasicIo is an interface for simple binary IO that was designed to have semantics and names similar to those of C style FILE* operations. Internally it already uses a FILE*, which can't be supplied nor accessed from the outside.

Sharing this FILE* with the outside world has the risk that the state of the FileIo object gets corrupted. (See how FileIo uses the pointer.) I (still) don't see what benefits it has.

#8 - 19 Dec 2010 22:48 - Adam Hooper

Maybe I can list a few more advantages. Allowing an interface for passing in a FILE* would:

- (as mentioned before) avoid a bizarre error case
- avoid the need to negotiate, lock, cache, etc. a network file twice (over NFS, for instance, this is dog-slow and error-prone; SSH/HTTP/FTP connections with FUSE cause a performance hit too)
- allow handling files outside of the filesystem, for instance tempfiles created by tmpfile() (which are often unlinked as soon as they're created)
- (this would take extra work ... just brainstorming here) allow the user to manipulate the FILE* before passing it in, say, by fast-forwarding to the part of a larger file where the Exif data begins.

#9 - 19 Dec 2010 23:36 - Andreas Huggel

Achieving all of this is not just a matter of adding a constructor to the existing class FileIo. Take a quick look at its implementation. E.g., FileIo happily closes and re-opens the file, possibly several times during the object's lifetime (needed to make the file open mode transparent to the user). It seeks to the beginning of the file. And it generally is one of the hairier parts of the Exiv2 code, having been tweaked for all sorts of weird issues on various platforms over the years. That also means it is difficult to regression-test.

In view of all this, I think it's advisable to implement a new BasicIo-derived class with the desired behavior from scratch rather than modifying the existing class FileIo.

#10 - 20 Dec 2010 14:27 - Robin Mills

Andreas Huggel wrote:

> > Maybe I should consider creating a FileIO object using FILE*
>
> Robin, what do you have in mind?
>
> Sharing this FILE* with the outside world has the risk that the state of the FileIo object gets corrupted. (See how FileIo uses the pointer.) I (still) don't see what benefits it has.

Well, if you don't see any benefit then I'm going to do nothing about this. However I'll keep this is mind as I move forward with my project. If a clear need arises, I'll implement this. I've been easily able to integrate exiv2 into my client's product. They open a FILE* and exiv2 opens another. However this hasn't caused any difficulty.

Bottom line: I'm not going to do anything about this.

#11 - 21 Aug 2015 19:12 - Robin Mills

- Status changed from New to Closed

- Assignee set to Robin Mills

- Target version set to 0.26

I'm going to close this issue. I don't believe anything needs to be done.

As a point of information, Exiv2 v0.25 provides remote I/O using curl/libssh and openssl. It doesn't use Fuse. If the user mounts a remote file using

Fuse, exiv2 would see this has an object in the file system and use FileIo. So fuse would be used and curl would be ignored.

As curl/libssh/openssl impose considerable build and library overhead, exiv2 has a default http protocol implementation which allows the user to read metadata from webservers. The implementation uses ByteRange to minimise network block reads of typically 1024 bytes. All versions of exiv2 v0.25 (whether the are linked to curl or not) can successfully execute requests such as this:

```
$ exiv2 -pv -g ExposureMode -g ExposureTime http://dev.exiv2.org/attachments/download/574/IMG_4810.JPG
0x829a Photo        ExposureTime                Rational    1  605/10
0xa402 Photo        ExposureMode                Short       1  1
$
```

#12 - 22 Aug 2015 16:13 - Robin Mills

- % Done changed from 0 to 100