GSOC 2013 KDE

v.1

Exiv2 "Cloud Ready" Project

Prepared by Nhu Dinh Tuan • April 15, 2013

Project Proposal

v.1

for John Cooper • Aug 15, 2012

1. Introduction

1.1. Personal Details

- Name: Nhu Dinh Tuan
- Email address: nhudinhtuan@gmail.com
- Phone number: +65 81791386
- Address:

  25 Prince George's Park

  Residence 6, BLK 27 #B1-09 [postcode: 118424]

  Singapore
- Location (Time Zone): UTC/GMT +8 (SGT - Singapore Time)
- IM Service and Username:
  - Skype: nhudinhtuan
  - Yahoo: nhudinhtuan1990@yahoo.com
  - Google: nhudinhtuan@gmail.com

1.2. Background and motivation

I am a third-year undergraduate student from the National University of Singapore. My major is Computer Science. I am proficient in C, C++, Objective C and Java. Web Technologies (HTTP(S) protocol, PHP, HTML5, Jquery, CSS3 ...) are also my strengths. I am a fast learner and have a passion to learn new things. As such, I am able to pick up new knowledge quickly.

Among the projects offered by KDE, I would like to work on Exiv2 "Cloud Ready" Project. This is because this project is related to cloud computing, which I am interested in. In addition, I have a good background knowledge in the subject area that is required for this project. More than that, I would like to contribute to open source community, in general, and to KDE, in particular. I also believe that I would improve my programming skills, get experience as well as learn new important things by doing this project.

About Exiv2, I have just known about it since two months ago. However, I've contacted with Robin quite early (almost 2 months), and we have discussed the project in details. Furthermore, I have already read the source code, made the prototype for *StdinIo*, *HttpIo* as well as a simple test harness. Hence, I am clear about the goal of the project, what I am going to do and I am confident to complete it on time.

2. Project Proposal

2.1. Scope and purpose

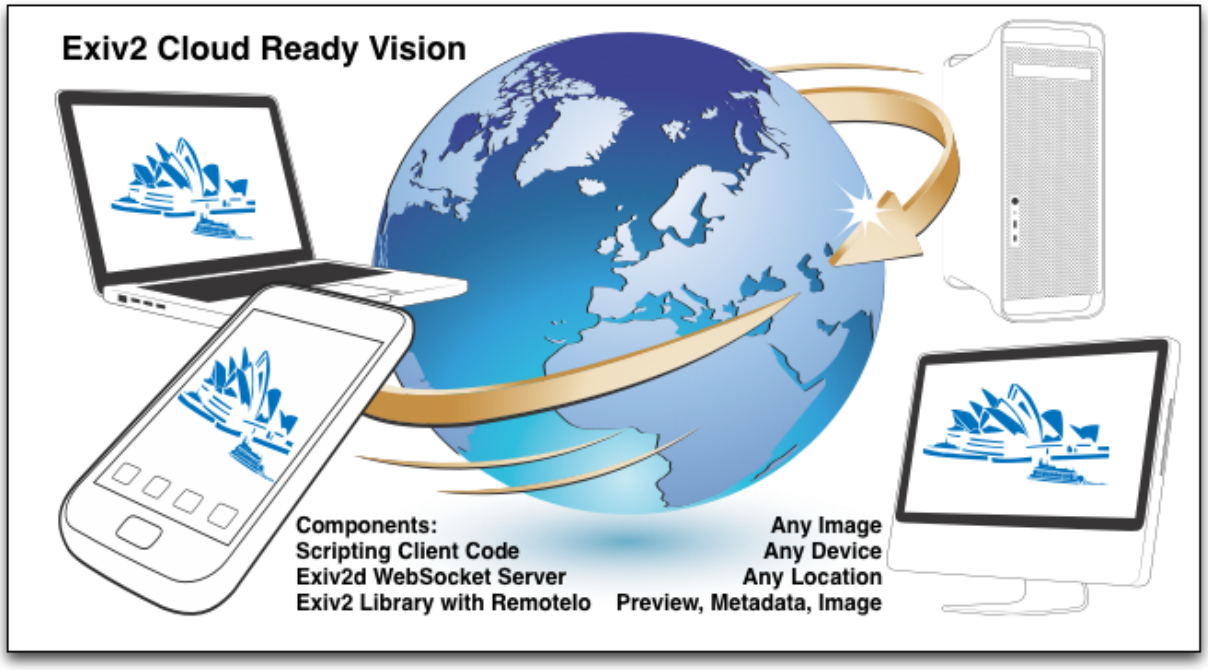


Fig.1. Exiv2 Cloud Ready Vision (created by Robin Mills)

Today we provide support files available on the file system. These files can be memory mapped if this feature is supported by the host OS. With the increasing interest in "cloud" computing, it's become ever more common for files to reside in remote locations which are not mapped to the file system. Very common cases today are ftp and http. This proposal is to support http, ftp and ssh for Exiv2. For example:

- exiv2 -pt http://clanmills.com/files/Robin.jpg
- exiv2 -pt ftp://username@password:/clanmills.com/Robin.jpg
- exiv2 -pt ssh://username@password:/clanmills.com/Robin.jpg

The implementation should provide bi-directional support (both read and write) with read-access being the first priority. It is necessary for the implementation to make efficient use of bandwidth, avoid adding a large library to the build dependencies as well as there is no dependency on platform frameworks such as .Net, Java, or Cocoa.

2.2. Architecture

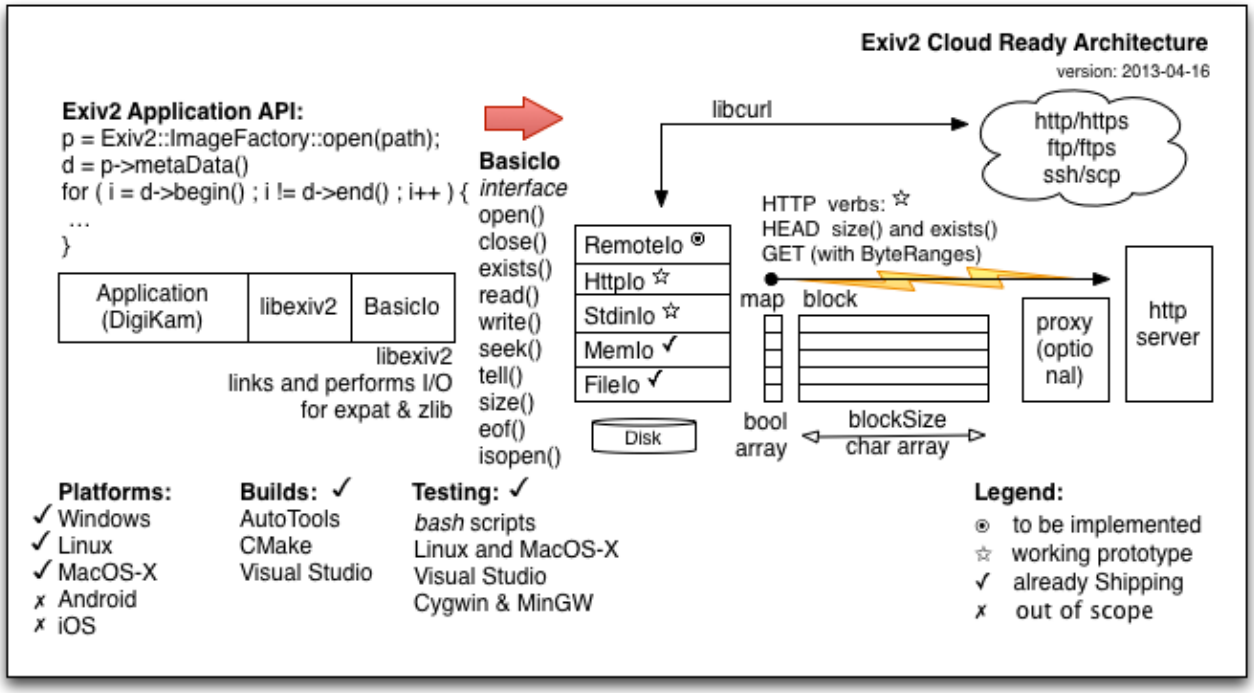


Fig. 2. Exiv2 Cloud Ready Architecture (created by Robin Mills)

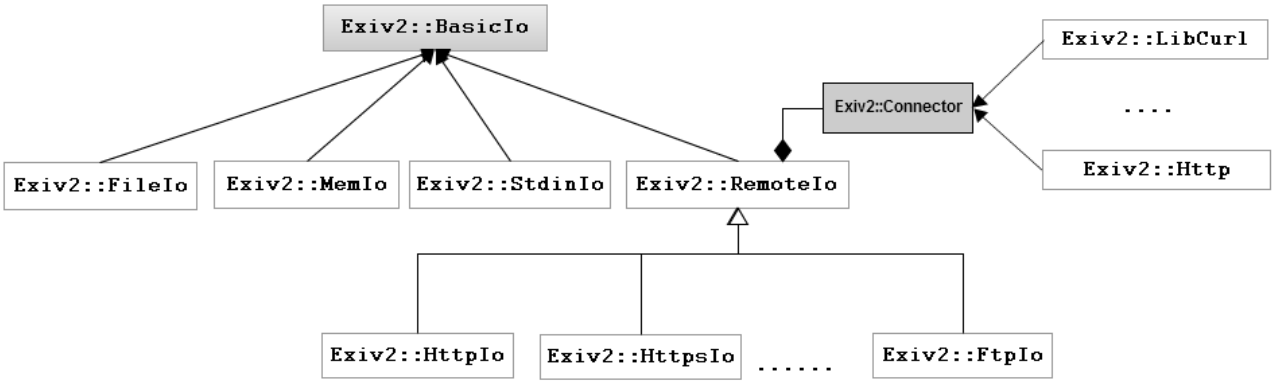


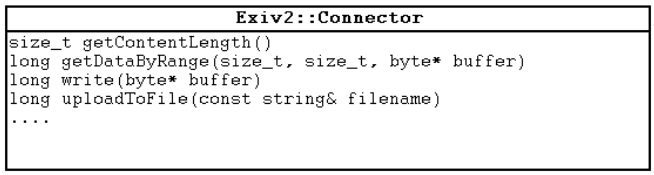
Fig. 3. Domain Model

- *StdinIo*: the class provides Stdin IO by implementing the *BasicIo* interface. This is just an addition feature which allows Exiv2 to accept the path "-" (meaning stdin).
- *Connector*: the interface for data transfer classes.
- *LibCurl*, *Http*: the classes for transferring the data via various protocols.
- *RemoteIo*: the class providing read/write access to remote files. It contains an *Connector* and is inherited by *HttpIo*, *HttpsIo*, *FtpIo* ... More details are discussed later in the implementation part.

Note: the *HttpIo* in the Fig. 2. is a little bit different from *HttpIo* in Fig. 3. The one in Fig. 2. is just the prototype which uses *Http*, implements the block map (the detail is mentioned in the implementation part) and it is not derived from RemoteIo.

2.3. Implementation

2.3.1 Connector Interface & data transfer classes



The *Connector* interface allows users to separate the definition of data transfer classes from their implementation. Hence, users are free to choose the implementation (libcurl, libferit, ...) which they prefer.

- *For read access: Connector* contains two abstract methods. One method is to get the content length (the size of the entity-body) of the remote files, another is to get the partial file contents by the ranges. These two methods are important to implement the block map (more details in 2.3.2).
- *For write access: Connector* contains a few abstract methods to allow upload a partial content or complete files. For some cases (HTTP PUT/POST), this of course requires that someone put a program or script on the server side that knows how to receive a stream from the *Connector*.

By default, I will provide the *Http* and *LibCurl* data transfer classes which implemented *Connector* interface. The conditional inclusions (#ifdef, #ifndef ...) are put in the code so that users can have a build switch between them.

- *Http*: This is implemented based on the file http.cpp. Robin has provided a 500 line file http.cpp which appears to be sufficient for http:/// read access. It's a thread safe C++ function which accepts and STL request container and it returns the server response and status. The file is a very light, cross-platform class for HTTP {GET|HEAD|PUT}. It's is used when *LibCurl* is not available
- *LibCurl*: This is implemented based on Curl library. Curl library is a free and easy-to-use client-side URL transfer library, supporting various protocols. Moreover, curl library is highly portable, it builds and works identically on numerous platforms.

2.3.2 RemoteIo and it's derived classes.

The simplest possible implementation of this proposal for exiv2 is to detect the protocol and uses a helper application such as curl or ssh. This implementation probably requires copying the complete file from the remote storage to a temporary file in the local file system. While such an implementation can be constructed quickly, this does not satisfy the project aim to make efficient use of bandwidth. In most image files, the meta-data is defined in the first 100k of the file, so the implementation should only read blocks on demand from the server and avoid copying the complete file. This gives ideas to implement the block map in *RemoteIo::Impl* (Pimpl idiom). It should allocate memory for the complete file and populate the memory with data "just in time" by maintaining a map of blocks which have been copied from the server.

Exiv2::RemoteIo
Connector* con; class Impl; Impl* p_;
....

Exiv2::RemoteIo::Impl
byte* data_; size_t size_; size_t blockSize_; bool* blocksRead_; long idx_; bool eof_; ....
....

The map is very simple – just an array of bools. When open() is called, get the size of the file, allocate memory for the complete file and an array of memory to hold a bool for every block. The blockSize can be any size >= 1 (defined in constructor).

```
int RemoteIo::open() {
    ....
    size_t size_ = con->getContentLength();
    size_t nBlocks = (p_->size_ + p_->blockSize_ - 1) / p_->blockSize_;
    p_->data_ = (byte*) std::malloc(nBlocks * p_->blockSize_);
    p_->idx_ = 0;
    p_->blocksRead_ = new bool[nBlocks];
    ::memset(p_->blocksRead_, false, nBlocks);
    ...
}
```

When a read is requested, *RemoteIo* should ensure the appropriate blocks have been requested, update the map, return data.

```
int RemoteIo::read(long rcount) {
    ....
    size_t lowBlock = p_->idx_ /p_->blockSize_;
    size_t highBlock = (p_->idx_ + rcount)/p_->blockSize_;
    while(blocksRead_[lowBlock] && lowBlock < highBlock) lowBlock++;
    while(blocksRead_[highBlock] && highBlock > lowBlock) highBlock--;
    p_->populateBlocks(con, lowBlock, highBlock);
    ...
}
```

The derived classes (*HttpIo*, *FtpIo*, *SSHIo*...) inherit the block map implementation from the base class, but they need to initialize the suitable connector object in their own constructor. For example, *FtpIo* needs to pass username & password to initialize the *Connector* while it's usually not required for *HttpIo*. The derived classes also override the methods of *RemoteIo* with their suitable implementation.

Moreover, it's necessary to add the a new method in the image factory (image.cpp) to sniff the path and return the correct new *BasicIo* object:

```
BasicIo::AutoPtr ImageFactory::createIo(const std::string& path) {
     if (path.compare("-") == 0) return BasicIo::AutoPtr(new StdinIo());
     else if (path.find("http://")) return BasicIo::AutoPtr(new HttpIo(path));
     else if (path.find("ftp://")) return BasicIo::AutoPtr(new FtpIo(path));
     else if ( path.find("file:///") return BasicIo::AutoPtr(new FileIo(path.substr(8)));
     .....
}
Image::AutoPtr ImageFactory::open(const std::string& path) {
     Image::AutoPtr image = open(ImageFactory::createIo(path));
     ....
}
```

2.4. Build and test strategy

The build and test should occur frequently enough that no errors can arise without noticing them and correcting them immediately. I will allocate at least 1.0 day/week of the project to cover build and test development on all supported environments. Most of the build work will come early in the project (we'll need to update each of the build environments to deal with libcurl before doing any work) while most of the test work will come later in the project once we have the classes working.

There are 3 build environments for Exiv2:

i. The "autotools" build uses the code in <exiv2>/config to create the configure script. When we do

```
$ make config
```

./configure (and some other files) are created.

ii. CMake "Cross Platform Make" uses the CMakeLists.txt in every directory to generate a build environment for a "Generator". Typical generators are "Unix Makefiles", "Eclipse CDT – Unix Makefiles", Xcode, and various editions of Visual Studio (see macro cm.bat).

iii. Visual Studio comes in two forms:

- msvc - support 32 bit builds on Visual Studio 2003/5/8. This environment requires the use to build expat and zlib in prescribed directories.
- msvc64 – supports 32 bit AND 64 bit builds on Visual Studio 2005/8/10/12.

Moreover, there are a couple of files: exv_conf.h/exv_msvc.h which control what is built in/out. exv_conf.h is generated by autotools. exv_msvc.h has to be edited by hand. We will have to add options for the following purposes:

- The prototype implements *StdinIo* is two ways and a build switch is provided to choose between deriving the class from *FileIo* or *MemIo*.
- We will use the same idea to cause *HttpIo* to be built into Exiv2 from *Http* or *LibCurl*. The code above is not aware of the implementation.
- The class *RemoteIo* will be based on *Libcurl* and subclasses for *Ftp*, *Http* and other protocols will be derived. In the case of *HttpIo*, we intend to provide an alternative implementation based on the *Http* function in our prototype. A build switch will be provided to allow the build engineer to choose how *HttpIo* is to be build. All future versions of Exiv2 will support *HttpIo* for reading.

There are a significant difficulties in dealing with and performing the build:

1. Availability of platforms and compilers.
2. Time to perform builds and run the test suite. The many editions of Visual Studio take time.

AWS EC2 (Elastic Compute Cloud) may be useful to address these concerns. We can create and configure cloud based machines to perform build and tests. The test suite (httpiotest.sh, ftpiotest.sh ...) operate by running programs using the library and comparing the output with validated reference files.

Robin has expressed a desire to have "build servers" which would build Exiv2 in response to commits to the repository. It is not a project aim to create a remote build system, however if time allows, this may be achieved.

3. Schedule

Time-period	Tasks	Deliverables
Week 1 [17 Jun - 23 Jun]	Build Libcurl in different environments.	Exiv2::LibCurl
Week 2 [24 Jun - 30 Jun]	HTTP/Read	Exiv2::RemoteIo, Exiv2::Connector, Exiv2::LibCurl*, Exiv2::Http, Exiv2::HttpIo
Week 3 [1 Jul - 7 Jul]	Build/Test and enhancement for HTTP/Read	
Week 4 [8 Jul - 14 Jul]	FTP/Read	Exiv2::Connector*, Exiv2::LibCurl*, Exiv2::FtpIo
Week 5 [15 Jul - 21 Jul]	Build/Test and enhancement for FTP/Read	
Week 6 [22 Jul - 28 Jul]	Documentation for mid-term evaluation	Documentation
Week 7 [29 Jul - 4 Aug] *August 2, 19:00 UTC: Mid-term evaluations deadline*	Research/Implementation for FTP/Write.	Exiv2::Connector*, Exiv2::LibCurl*, Exiv2::FtpIo*
Week 8 [5 Aug - 11 Aug]	Build/Test & enhancement for FTP/Write	
Week 9 [12 Aug - 18 Aug]	Research/Implementation about WebDav.	Exiv2::WebDav
Week 10 [19 Aug - 25 Aug]	Build/Test & enhancement for WebDav	
Week 11 [26 Aug - 1 Sep]	HTTP/Write (PUT|POST)	Exiv2::Http*, Exiv2::LibCurl*, Exiv2::HttpIo*
Week 12 [2 Sep - 8 Sep]	Build/Test & enhancement for HTTP/Write	
Week 13 [9 Sep - 15 Sep]	Build/Test & enhancement	
Week 14 [16 Sep - 22 Sep] *September 27, 19:00 UTC: Final evaluation deadline*	Documentation for Final evaluation.	Documentation

Note:

* means “more methods may be added to the class”.

4. Prototype Investigation

I made the prototypes for *StdinIo* and *HttpIo*. I learned about Pimpl Idiom through reading the source code.

It's used in my prototypes to minimize coupling via the separation of interface and implementation and then implementation hiding. I investigated about the various protocols: HTTP (using HEAD to get the content-length, how to deal with servers don't support byte ranges ...), FTP (how to get the content length, the partial of files ...). Moreover, I tried working with LibCurl, implemented a simple program to work with http, https, ftp.

The following is the performance of block map in *Httpio* prototype (when MISS occurs, connect to the server and load the missing data-blocks):

```
$./exifprint http://exiv2.nuditu.com/httpio0.jpg
    BlockSize = 512
    Content-Length = 32764
    MISS = 2 times
    Total bytes loaded: 8192 (25%)
    ---------------------------------
    BlockSize = 1024
    Content-Length = 32764
    MISS = 2 times
    Total bytes loaded: 8192 (25%)

$./exifprint http://exiv2.nuditu.com/httpio1.jpg
    BlockSize = 512
    Content-Length = 496679
    MISS = 6 times
    Total bytes loaded: 19968 (4%)
    ---------------------------------
    BlockSize = 1024
    Content-Length = 496679
    MISS = 6 times
    Total bytes loaded: 20480 (4.1%)

$./exifprint http://exiv2.nuditu.com/httpio2.jpg
    BlockSize = 512
    Content-Length = 5611406
    MISS = 6 times
    Total bytes loaded: 20480 (0.36%)
    ---------------------------------
    BlockSize = 1024
    Content-Length = 5611406
    MISS = 5 times
    Total bytes loaded: 20480 (0.36%)

$./exiv2 -u -pa http://clanmills.com/2010/PyCon/RobinPyCon2010.mov
    BlockSize = 512
    Content-Length = 40251022
    MISS = 11 times
    Total bytes loaded: 5774 (0.014%)
    ---------------------------------
    BlockSize = 1024
    Content-Length = 40251022
    MISS = 10 times
    Total bytes loaded: 10894 (0.027%)
```