FILE OPENING

```cpp
typedef std::auto_ptr<Image> AutoPtr;
typedef std::auto_ptr<BasicIo> AutoPtr;
```

```cpp
Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(filePathStr);


Image::AutoPtr ImageFactory::open(const std::string& path, bool useCurl)
{
    Image::AutoPtr image = open(ImageFactory::createIo(path, useCurl));
    (…)
    return image;
}
```

```cpp
            return BasicIo::AutoPtr(new FileIo(path));
```

```cpp
class FileIo : public BasicIo {
 public:
 //(…) constructor does not open the file
 FileIo(const std::string& path)
    : p_(new Impl(path))


    Impl(const std::string& path)
     : path_(path), (…), pMappedArea_(0), mappedLength_(0), isMalloced_(false),
    isWriteable_(false)
```

```cpp
Image::AutoPtr ImageFactory::open(BasicIo::AutoPtr io)
{
    (…)
    for (unsigned int i = 0; registry[i].imageType_ != ImageType::none; ++i) {
        if (registry[i].isThisType_(*io, false)) {
            return registry[i].newInstance_(io, false);
        }
    }
    return Image::AutoPtr();
}
```

```cpp
struct Registry {
        //! Comparison operator to compare a Registry structure with an image type
        bool operator==(const int& imageType) const { return imageType == imageType_; }

        // DATA
        int             imageType_;
        NewInstanceFct  newInstance_;
        IsThisTypeFct   isThisType_;
        AccessMode      exifSupport_;
        AccessMode      iptcSupport_;
        AccessMode      xmpSupport_;
        AccessMode      commentSupport_;
    };
```

```cpp
Image::AutoPtr newCr2Instance(BasicIo::AutoPtr io, bool create)
{
    Image::AutoPtr image(new Cr2Image(io, create));
    if (!image->good()) {
        image.reset();
    }
    return image;
}
            Cr2Image::Cr2Image(BasicIo::AutoPtr io, bool /*create*/)
                : Image(ImageType::cr2, mdExif | mdIptc | mdXmp, io)
```

```cpp
const Registry registry[] = {
//image type        creation fct       type check   Exif mode     IPTC mode     XMP mode      Comment mode
//---------------   ---------------    ----------   ---------     ---------     ---------     ------------
{ ImageType::jpeg, newJpegInstance, isJpegType,  amReadWrite,  amReadWrite,  amReadWrite,  amReadWrite },
{ ImageType::exv,  newExvInstance,  isExvType,   amReadWrite,  amReadWrite,  amReadWrite,  amReadWrite },
{ ImageType::cr2,  newCr2Instance,  isCr2Type,   amReadWrite,  amReadWrite,  amReadWrite,  amNone      },
{ ImageType::crw,  newCrwInstance,  isCrwType,   amReadWrite,  amNone,       amNone,       amReadWrite },
{ ImageType::mrw,  newMrwInstance,  isMrwType,   amRead,       amRead,       amRead,       amNone      },
{ ImageType::tiff, newTiffInstance, isTiffType,  amReadWrite,  amReadWrite,  amReadWrite,  amNone      },
{ ImageType::webp, newWebPInstance, isWebPType,  amReadWrite,  amNone,       amReadWrite,  amNone      },
{ ImageType::dng,  newTiffInstance, isTiffType,  amReadWrite,  amReadWrite,  amReadWrite,  amNone      },
{ ImageType::nef,  newTiffInstance, isTiffType,  amReadWrite,  amReadWrite,  amReadWrite,  amNone      },
{ ImageType::pef,  newTiffInstance, isTiffType,  amReadWrite,  amReadWrite,  amReadWrite,  amNone      },
(…)
```

Reading Metadata in Cr2Image

```cpp
Exiv2::Image::AutoPtr image = Exiv2::ImageFactory::open(filePathStr);
image->readMetadata();
```

```cpp
void Cr2Image::readMetadata()
    { (…)
        clearMetadata();
        ByteOrder bo = Cr2Parser::decode(exifData_,
                                         iptcData_,
                                         xmpData_,
                                         io_->mmap(),
                                         io_->size());

        setByteOrder(bo);
    }
```

Protected attribute inherited from Exiv2::Image class

```cpp
protected:
    BasicIo::AutoPtr  io_;            //!< Image data IO pointer
    ExifData          exifData_;      //!< Exif data container
    IptcData          iptcData_;      //!< IPTC data container
    XmpData           xmpData_;       //!< XMP data container
(…)
```

```cpp
    { Cr2Header cr2Header;
      ByteOrder TiffParserWorker::decode(exifData,
                                         (…)
                                         &cr2Header);}
```

```cpp
{// Create standard TIFF header if necessary
 std::auto_ptr<TiffHeaderBase> ph;
   if (!pHeader) {
     ph = std::auto_ptr<TiffHeaderBase>(new TiffHeader);
     pHeader = ph.get();
   }
TiffComponent::AutoPtr rootDir = parse(pData, size, root, pHeader);
    if (0 != rootDir.get()) {
        TiffDecoder decoder(exifData, (…)
                        rootDir.get(),
                        findDecoderFct);
        rootDir->accept(decoder);
     }
    return pHeader->byteOrder();
  }
```

> Which of the TiffComponent derived classes
> are we dealing with in the case of Cr2Image

```cpp
class TiffDecoder : public TiffVisitor {
    public:
        TiffDecoder(
            ExifData&          exifData,
            IptcData&          iptcData,
            XmpData&           xmpData,
            TiffComponent* const pRoot,
            FindDecoderFct     findDecoderFct
        )
(…)
{ // Find camera make
    TiffFinder finder(0x010f, ifd0Id);
    pRoot_->accept(finder);
    TiffEntryBase* te = dynamic_cast<TiffEntryBase*>(finder.result());
    if (te && te->pValue()) {
        make_ = te->pValue()->toString();
    }
}
```

```cpp
TiffFinder(uint16_t tag, IfdId group)
     : tag_(tag), group_(group), tiffComponent_(0) {}
```

```cpp
void TiffComponent::accept(TiffVisitor& visitor)
{
    if (visitor.go(TiffVisitor::geTraverse)) doAccept(visitor);
}
```

```cpp
virtual void doAccept(TiffVisitor& visitor) =0;
```

Which doAccept() is used depends on which TiffComponent-derived concrete class is used.
Which one is it then in the case of Cr2Image?

Reading EXIF data

`Exiv2::ExifData &exifData = image->exifData();`

```cpp
class ExifData {

    private:
        typedef std::list<Exifdatum> ExifMetadata;
        ExifMetadata exifMetadata_;
    public:
/// Raphael: Implicitly-declared default constructor.
        //! ExifMetadata iterator type
        typedef ExifMetadata::iterator iterator;
        typedef ExifMetadata::const_iterator const_iterator;
// Returns a reference to the %Exifdatum that is associated with a
// particular \em key.
Exifdatum& ExifData::operator[](const std::string& key)
    {
        ExifKey exifKey(key);
        iterator pos = findKey(exifKey);
        if (pos == end()) {
            add(Exifdatum(exifKey));
            pos = findKey(exifKey);
                            exifMetadata_.push_back(exifdatum);
        }
        return *pos;
    }
iterator begin() { return exifMetadata_.begin(); }
iterator end() { return exifMetadata_.end(); }
ExifData::const_iterator findKey(const ExifKey& key) const
    {
        return std::find_if(exifMetadata_.begin(), exifMetadata_.end(),
                            FindExifdatumByKey(key.key()));
    }
```

```cpp
class Exifdatum : public Metadatum {
(…)
private:
        ExifKey::AutoPtr key_;
        Value::AutoPtr  value_;
    };
std::string Exifdatum::key() const
    { return key_.get() == 0 ? "" : key_->key();}
const char* Exifdatum::familyName() const
std::string Exifdatum::groupName() const
std::string Exifdatum::tagName() const
```

```cpp
struct ExifKey::Impl {
        (…)
        void decomposeKey(const std::string& key);
        // DATA
        static const char* familyName_;    //!< "Exif"

        const TagInfo* tagInfo_;            //!< Tag info
        uint16_t tag_;                      //!< Tag value
        IfdId ifdId_;                       //!< The IFD associated with this tag
        int idx_;                           //!< Unique id of the Exif key in the image
        std::string groupName_;             //!< The group name
        std::string key_;                   //!< %Key
    };
```

```cpp
ExifKey::ExifKey(const std::string& key)
        : p_(new Impl)
    {
        p_->decomposeKey(key);
    }

void ExifKey::Impl::decomposeKey(const std::string& key)
(…)
// tagName() translates hex tag name (0xabcd) to a real tag name if there is one
        key_ = familyName + "." + groupName + "." + tagName();

std::string ExifKey::Impl::tagName() const
    {
        if (tagInfo_ != 0 && tagInfo_->tag_ != 0xffff) {
            return tagInfo_->name_;
        }
        std::ostringstream os;
        os << "0x" << std::setw(4) << std::setfill('0') << std::right
            << std::hex << tag_;
        return os.str();
    }
```