FILE OPENING

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

```
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;
    }
```

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

```
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)
```

```
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();
    }
```

```
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_;
    };
```

```
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)
```

```
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 EXIF data

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

```
class ExifData {
    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);
        }
        return *pos;
    }
```

```
private:
        // DATA
        ExifMetadata exifMetadata_;
```

```
typedef std::list<Exifdatum> ExifMetadata;
```