Library examples

Example 1: exifprint.cpp

This is a very simple program to read and print the Exif metadata of an image. Go to **Example2** to see how the output looks like.

```
// ***************************************************************** -*- C++ -*-
// exifprint.cpp, $Rev: 3090 $
// Sample program to print the Exif metadata of an image

#include <exiv2/exiv2.hpp (doc/exiv2_8hpp.html)>

#include <iostream>
#include <iomanip>
#include <cassert>

int main(int argc, char* const argv[])
try {

    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " file\n";
        return 1;
    }

    Exiv2::Image::AutoPtr (doc/classExiv2_1_1Image.html#a89ad3ffe7a4e8a943d267d77843415fb)
       image = Exiv2::ImageFactory::open
       (doc/classExiv2_1_1ImageFactory.html#aba929c4ca4a71625d12bcb97bcc28161)(argv[1]);
    assert(image.get() != 0);
    image->readMetadata();

    Exiv2::ExifData (doc/classExiv2_1_1ExifData.html) &exifData = image->exifData();
    if (exifData.empty (doc/classExiv2_1_1ExifData.html#a4993c68fbb50731014c307852875c731)()) {
        std::string error(argv[1]);
        error += ": No Exif data found in the file";
        throw Exiv2::Error (doc/namespaceExiv2.html#accd3e49cafe9db52c1e0e6f648753cae)(1,
       error);
    }
    Exiv2::ExifData::const_iterator
       (doc/classExiv2_1_1ExifData.html#a2b8ac7a474d6527c0f3f6a0a9cebef77) end = exifData.end
       (doc/classExiv2_1_1ExifData.html#a9c15177b03489e3d4bb81e9acc1165fe)();
    for (Exiv2::ExifData::const_iterator
       (doc/classExiv2_1_1ExifData.html#a2b8ac7a474d6527c0f3f6a0a9cebef77) i = exifData.begin
       (doc/classExiv2_1_1ExifData.html#a53bce2980ee060fc2da5fe6751f51db9)(); i != end; ++i) {
        const char* tn = i->typeName();
        std::cout << std::setw(44) << std::setfill(' ') << std::left
                  << i->key() << " "
                  << "0x" << std::setw(4) << std::setfill('0') << std::right
                  << std::hex << i->tag() << " "
                  << std::setw(9) << std::setfill(' ') << std::left
                  << (tn ? tn : "Unknown") << " "
                  << std::dec << std::setw(3)
                  << std::setfill(' ') << std::right
                  << i->count() << "  "
                  << std::dec << i->value()
                  << "\n";
    }

    return 0;
}
//catch (std::exception& e) {
//catch (Exiv2::AnyError& e) {
catch (Exiv2::Error (doc/classExiv2_1_1BasicError.html)& e) {
    std::cout << "Caught Exiv2 exception '" << e.what
       (doc/classExiv2_1_1BasicError.html#a72e9f29e45d6f59125fa3de232641504)() << "'\n";
    return -1;
}
```

Example 2: addmoddel.cpp

Sample usage of high-level Exiv2 library calls to add, modify and delete Exif metadata.

```
// ***************************************************************** -*- C++ -*-
// addmoddel.cpp, $Rev: 3353 $
// Sample program showing how to add, modify and delete Exif metadata.

#include <exiv2/exiv2.hpp (doc/exiv2_8hpp.html)>

#include <iostream>
#include <iomanip>
#include <cassert>

int main(int argc, char* const argv[])
try {
    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " file\n";
        return 1;
    }
    std::string file(argv[1]);

    // Container for exif metadata. This is an example of creating
    // exif metadata from scratch. If you want to add, modify, delete
    // metadata that exists in an image, start with ImageFactory::open
    Exiv2::ExifData (doc/classExiv2_1_1ExifData.html) exifData;

    // *************************************************************************
    // Add to the Exif data

    // This is the quickest way to add (simple) Exif data. If a metadatum for
    // a given key already exists, its value is overwritten. Otherwise a new
    // tag is added.
    exifData["Exif.Image.Model"] = "Test 1";                     // AsciiValue
    exifData["Exif.Image.SamplesPerPixel"] = uint16_t(162);      // UShortValue
    exifData["Exif.Image.XResolution"] = int32_t(-2);            // LongValue
    exifData["Exif.Image.YResolution"] = Exiv2::Rational
       (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(-2, 3); // RationalValue
    std::cout << "Added a few tags the quick way.\n";

    // Create a ASCII string value (note the use of create)
    Exiv2::Value::AutoPtr (doc/classExiv2_1_1Value.html#a0f62e585b82c97738858b743e60dff21) v =
       Exiv2::Value::create (doc/classExiv2_1_1Value.html#ad6ff043921cd1a5c399a9a4fc8257006)
       (Exiv2::asciiString
       (doc/namespaceExiv2.html#a5153319711f35fe81cbc13f4b852450ca773cf6dde5caaabb3dcf9fb161fa7
       dfd));
    // Set the value to a string
    v->read("1999:12:31 23:59:59");
    // Add the value together with its key to the Exif data container
    Exiv2::ExifKey (doc/classExiv2_1_1ExifKey.html) key("Exif.Photo.DateTimeOriginal");
    exifData.add (doc/classExiv2_1_1ExifData.html#a91d231cd1b9fefc311c5166e30ab66eb)(key,
       v.get());
    std::cout << "Added key \"" << key << "\", value \"" << *v << "\"\n";

    // Now create a more interesting value (without using the create method)
    Exiv2::URationalValue::AutoPtr
       (doc/classExiv2_1_1ValueType.html#a0c76c512468a47f6eac463f4af278a14) rv(new
       Exiv2::URationalValue (doc/classExiv2_1_1ValueType.html));
    // Set two rational components from a string
    rv->read("1/2 1/3");
    // Add more elements through the extended interface of rational value
    rv->value_.push_back(std::make_pair(2,3));
    rv->value_.push_back(std::make_pair(3,4));
    // Add the key and value pair to the Exif data
    key = Exiv2::ExifKey (doc/classExiv2_1_1ExifKey.html)("Exif.Image.PrimaryChromaticities");
    exifData.add (doc/classExiv2_1_1ExifData.html#a91d231cd1b9fefc311c5166e30ab66eb)(key,
       rv.get());
    std::cout << "Added key \"" << key << "\", value \"" << *rv << "\"\n";

    // *************************************************************************
    // Modify Exif data

    // Since we know that the metadatum exists (or we don't mind creating a new
    // tag if it doesn't), we can simply do this:
    Exiv2::Exifdatum (doc/classExiv2_1_1Exifdatum.html)& tag =
       exifData["Exif.Photo.DateTimeOriginal"];
    std::string date = tag.toString
       (doc/classExiv2_1_1Exifdatum.html#a73d1e5346411c2adf520fec405f2e536)();
    date.replace(0, 4, "2000");
    tag.setValue(date);
    std::cout << "Modified key \"" << tag.key
       (doc/classExiv2_1_1Exifdatum.html#a6651602de3d217dd622d33ab67289c11)()
             << "\", new value \"" << tag.value
       (doc/classExiv2_1_1Exifdatum.html#ad4a621c1399e02648f1fb1fb550e7a53)() << "\"\n";

    // Alternatively, we can use findKey()
    key = Exiv2::ExifKey (doc/classExiv2_1_1ExifKey.html)("Exif.Image.PrimaryChromaticities");
    Exiv2::ExifData::iterator
       (doc/classExiv2_1_1ExifData.html#a02e2a2acb4cfeb0f7755c1a45f94106f) pos =
       exifData.findKey (doc/classExiv2_1_1ExifData.html#a96c38cbd300ebdfa05f849864b380690)
       (key);
    if (pos == exifData.end (doc/classExiv2_1_1ExifData.html#a9c15177b03489e3d4bb81e9acc1165fe)
       ()) throw Exiv2::Error (doc/classExiv2_1_1BasicError.html)(1, "Key not found");
    // Get a pointer to a copy of the value
    v = pos->getValue();
    // Downcast the Value pointer to its actual type
    Exiv2::URationalValue (doc/classExiv2_1_1ValueType.html)* prv =
       dynamic_cast<Exiv2::URationalValue (doc/classExiv2_1_1ValueType.html)*>(v.release());
    if (prv == 0) throw Exiv2::Error
       (doc/namespaceExiv2.html#accd3e49cafe9db52c1e0e6f648753cae)(1, "Downcast failed");
    rv = Exiv2::URationalValue::AutoPtr
       (doc/classExiv2_1_1ValueType.html#a0c76c512468a47f6eac463f4af278a14)(prv);
    // Modify the value directly through the interface of URationalValue
    rv->value_[2] = std::make_pair(88,77);
    // Copy the modified value back to the metadatum
    pos->setValue(rv.get());
    std::cout << "Modified key \"" << key
             << "\", new value \"" << pos->value() << "\"\n";

    // *************************************************************************
    // Delete metadata from the Exif data container

    // Delete the metadatum at iterator position pos
    key = Exiv2::ExifKey (doc/classExiv2_1_1ExifKey.html)("Exif.Image.PrimaryChromaticities");
    pos = exifData.findKey (doc/classExiv2_1_1ExifData.html#a96c38cbd300ebdfa05f849864b380690)
       (key);
    if (pos == exifData.end (doc/classExiv2_1_1ExifData.html#a9c15177b03489e3d4bb81e9acc1165fe)
       ()) throw Exiv2::Error (doc/classExiv2_1_1BasicError.html)(1, "Key not found");
    exifData.erase (doc/classExiv2_1_1ExifData.html#a710a66ca8be51192c15729c541b72fb5)(pos);
    std::cout << "Deleted key \"" << key << "\"\n";

    // *************************************************************************
    // Finally, write the remaining Exif data to the image file
    Exiv2::Image::AutoPtr (doc/classExiv2_1_1Image.html#a89ad3ffe7a4e8a943d267d77843415fb)
       image = Exiv2::ImageFactory::open
       (doc/classExiv2_1_1ImageFactory.html#aba929c4ca4a71625d12bcb97bcc28161)(file);
    assert(image.get() != 0);

    image->setExifData(exifData);
    image->writeMetadata();

    return 0;
}
catch (Exiv2::AnyError (doc/classExiv2_1_1AnyError.html)& e) {
    std::cout << "Caught Exiv2 exception '" << e << "'\n";
    return -1;
}
```

Using the print function from Example1 shows the following Exif tags in the image. Note the tag `Exif.Image.ExifTag` : It is required by the Exif standard because the metadata contains an `Exif.Photo.*` tag and is automatically added by Exiv2 to ensure that the Exif structure is valid.

```
$ exifprint img_2158.jpg
Exif.Image.Model                  0x0110 Ascii       7  Test 1
Exif.Image.SamplesPerPixel        0x0115 Short       1  162
Exif.Image.XResolution            0x011a SLong       1  -2
Exif.Image.YResolution            0x011b SRational   1  -2/3
Exif.Image.ExifTag                0x8769 Long        1  89
Exif.Photo.DateTimeOriginal       0x9003 Ascii      20  2000:12:31 23:59:59
```

Image with the Exif metadata from example 2

Example 3: iptcprint.cpp

This is a very simple program to read and print the IPTC metadata of an image.

```
// ***************************************************************** -*- C++ -*-
// iptcprint.cpp, $Rev: 3090 $
// Sample program to print the IPTC metadata of an image

#include <exiv2/exiv2.hpp (doc/exiv2_8hpp.html)>

#include <iostream>
#include <iomanip>
#include <cassert>
```

```
int main(int argc, char* const argv[])
try {

    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " file\n";
        return 1;
    }

    Exiv2::Image::AutoPtr (doc/classExiv2_1_1Image.html#a89ad3ffe7a4e8a943d267d77843415fb)
       image = Exiv2::ImageFactory::open
       (doc/classExiv2_1_1ImageFactory.html#aba929c4ca4a71625d12bcb97bcc28161)(argv[1]);
    assert (image.get() != 0);
    image->readMetadata();

    Exiv2::IptcData (doc/classExiv2_1_1IptcData.html) &iptcData = image->iptcData();
    if (iptcData.empty (doc/classExiv2_1_1IptcData.html#afda626e27ebecd599005c68022db9c1c)()) {
        std::string error(argv[1]);
        error += ": No IPTC data found in the file";
        throw Exiv2::Error (doc/namespaceExiv2.html#accd3e49cafe9db52c1e0e6f648753cae)(1,
       error);
    }

    Exiv2::IptcData::iterator
       (doc/classExiv2_1_1IptcData.html#a0d53776cd2f36e63fff78c8f142a7caf) end = iptcData.end
       (doc/classExiv2_1_1IptcData.html#a6753e8a713ab2b42a3bdc7b3d9eab401)();
    for (Exiv2::IptcData::iterator
       (doc/classExiv2_1_1IptcData.html#a0d53776cd2f36e63fff78c8f142a7caf) md = iptcData.begin
       (doc/classExiv2_1_1IptcData.html#a03385c128b29d262ade837093fddc0d2)(); md != end; ++md)
       {
        std::cout << std::setw(44) << std::setfill(' ') << std::left
                  << md->key() << " "
                  << "0x" << std::setw(4) << std::setfill('0') << std::right
                  << std::hex << md->tag() << " "
                  << std::setw(9) << std::setfill(' ') << std::left
                  << md->typeName() << " "
                  << std::dec << std::setw(3)
                  << std::setfill(' ') << std::right
                  << md->count() << "  "
                  << std::dec << md->value()
                  << std::endl;
    }

    return 0;
}
catch (Exiv2::AnyError (doc/classExiv2_1_1AnyError.html)& e) {
    std::cout << "Caught Exiv2 exception '" << e << "'\n";
    return -1;
}
```

```
$ iptcprint smiley1.jpg
Iptc.Application2.Headline              0x0069 String     17  The headline I am
Iptc.Application2.Keywords              0x0019 String     19  Yet another keyword
Iptc.Application2.DateCreated           0x0037 Date        8  2004-08-03
Iptc.Application2.Urgency               0x000a String      5  very!
Iptc.Envelope.ModelVersion              0x0000 Short       1  42
Iptc.Envelope.TimeSent                  0x0050 Time       11  14:41:00-05:00
Iptc.Application2.RasterizedCaption     0x007d Undefined   8  230 42 34 2 90 84 23 146
Iptc.0x0009.0x0001                      0x0001 String      9  Who am I?
```

Image with the IPTC data from examples 3 & 4

Example 4: iptceasy.cpp

This shows the quickest way to access, set or modify IPTC metadata, which is similar to how `std::map` works. The sample program writes the IPTC data to a file. Example 3, above, has the image with this IPTC data.

```
// ***************************************************************** -*- C++ -*-
// iptceasy.cpp, $Rev: 3090 $
// The quickest way to access, set or modify IPTC metadata.

#include <exiv2/exiv2.hpp (doc/exiv2_8hpp.html)>

#include <iostream>
#include <iomanip>
#include <cassert>

int main(int argc, char* const argv[])
try {
    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " file\n";
        return 1;
    }
    std::string file(argv[1]);

    Exiv2::IptcData (doc/classExiv2_1_1IptcData.html) iptcData;

    iptcData["Iptc.Application2.Headline"] = "The headline I am";
    iptcData["Iptc.Application2.Keywords"] = "Yet another keyword";
    iptcData["Iptc.Application2.DateCreated"] = "2004-8-3";
    iptcData["Iptc.Application2.Urgency"] = uint16_t(1);
    iptcData["Iptc.Envelope.ModelVersion"] = 42;
    iptcData["Iptc.Envelope.TimeSent"] = "14:41:0-05:00";
    iptcData["Iptc.Application2.RasterizedCaption"] = "230 42 34 2 90 84 23 146";
    iptcData["Iptc.0x0009.0x0001"] = "Who am I?";

    Exiv2::StringValue (doc/classExiv2_1_1StringValue.html) value;
    value.read (doc/classExiv2_1_1StringValueBase.html#a6882ba90138a30fcf2123c74f928a75e)
       ("very!");
    iptcData["Iptc.Application2.Urgency"] = value;

    std::cout << "Time sent: " << iptcData["Iptc.Envelope.TimeSent"] << "\n";

    // Open image file
    Exiv2::Image::AutoPtr (doc/classExiv2_1_1Image.html#a89ad3ffe7a4e8a943d267d77843415fb)
       image = Exiv2::ImageFactory::open
       (doc/classExiv2_1_1ImageFactory.html#aba929c4ca4a71625d12bcb97bcc28161)(file);
    assert (image.get() != 0);

    // Set IPTC data and write it to the file
    image->setIptcData(iptcData);
    image->writeMetadata();
    return 0;
}
catch (Exiv2::AnyError (doc/classExiv2_1_1AnyError.html)& e) {
    std::cout << "Caught Exiv2 exception '" << e << "'\n";
    return -1;
}
```

Example 5: xmpsample.cpp

Sample (test) usage of high level XMP classes. This example shows various aspects of setting XMP metadata, including complex types. See also Example 2: addmoddel.cpp

```
// ***************************************************************** -*- C++ -*-
// xmpsample.cpp, $Rev: 3090 $
// Sample/test for high level XMP classes. See also addmoddel.cpp

#include <exiv2/exiv2.hpp (doc/exiv2_8hpp.html)>

#include <string>
#include <iostream>
#include <iomanip>
#include <cassert>
#include <cmath>

bool isEqual(float a, float b)
{
    double d = std::fabs(a - b);
    return d < 0.00001;
}

int main()
try {
    // The XMP property container
    Exiv2::XmpData (doc/classExiv2_1_1XmpData.html) xmpData;

    // -------------------------------------------------------------------------
    // Teaser: Setting XMP properties doesn't get much easier than this:

    xmpData["Xmp.dc.source"]  = "xmpsample.cpp";    // a simple text value
    xmpData["Xmp.dc.subject"] = "Palmtree";         // an array item
    xmpData["Xmp.dc.subject"] = "Rubbertree";       // add a 2nd array item
    // a language alternative with two entries and without default
    xmpData["Xmp.dc.title"]   = "lang=de-DE Sonnenuntergang am Strand";
    xmpData["Xmp.dc.title"]   = "lang=en-US Sunset on the beach";

    // -------------------------------------------------------------------------
    // Any properties can be set provided the namespace is known. Values of any
    // type can be assigned to an Xmpdatum, if they have an output operator. The
    // default XMP value type for unknown properties is a simple text value.

    xmpData["Xmp.dc.one"]     = -1;
    xmpData["Xmp.dc.two"]     = 3.1415;
    xmpData["Xmp.dc.three"]   = Exiv2::Rational
       (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(5, 7);
    xmpData["Xmp.dc.four"]    = uint16_t(255);
    xmpData["Xmp.dc.five"]    = int32_t(256);
    xmpData["Xmp.dc.six"]     = false;

    // In addition, there is a dedicated assignment operator for Exiv2::Value
    Exiv2::XmpTextValue (doc/classExiv2_1_1XmpTextValue.html) val("Seven");
    xmpData["Xmp.dc.seven"]   = val;
    xmpData["Xmp.dc.eight"]   = true;

    // Extracting values
    assert(xmpData["Xmp.dc.one"].toLong() == -1);
    assert(xmpData["Xmp.dc.one"].value().ok());
```

```
const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv1 = xmpData["Xmp.dc.one"].value();
assert(isEqual(getv1.toFloat
   (doc/classExiv2_1_1Value.html#a22d257caa6c1ffe6416ce02de7bd8c1c)(), -1));
assert(getv1.ok (doc/classExiv2_1_1Value.html#a161550b3ef31b3a14b1d75149ba9ba71)());
assert(getv1.toRational (doc/classExiv2_1_1Value.html#a595a4cb549bec8c19d290ca3e95a2678)()
   == Exiv2::Rational (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(-1, 1));
assert(getv1.ok (doc/classExiv2_1_1Value.html#a161550b3ef31b3a14b1d75149ba9ba71)());

const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv2 = xmpData["Xmp.dc.two"].value();
assert(isEqual(getv2.toFloat(), 3.1415f));
assert(getv2.ok());
assert(getv2.toLong() == 3);
assert(getv2.ok());
Exiv2::Rational (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30) R =
   getv2.toRational();
assert(getv2.ok());
assert(isEqual(static_cast<float>(R.first) / R.second, 3.1415f ));

const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv3 = xmpData["Xmp.dc.three"].value();
assert(isEqual(getv3.toFloat
   (doc/classExiv2_1_1Value.html#a22d257caa6c1ffe6416ce02de7bd8c1c)(), 5.0f/7.0f));
assert(getv3.ok (doc/classExiv2_1_1Value.html#a161550b3ef31b3a14b1d75149ba9ba71)());
assert(getv3.toLong (doc/classExiv2_1_1Value.html#a4530a3fc3e2305cf994de5476f46f953)() ==
   0);  // long(5.0 / 7.0)
assert(getv3.ok (doc/classExiv2_1_1Value.html#a161550b3ef31b3a14b1d75149ba9ba71)());
assert(getv3.toRational (doc/classExiv2_1_1Value.html#a595a4cb549bec8c19d290ca3e95a2678)()
   == Exiv2::Rational (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(5, 7));
assert(getv3.ok (doc/classExiv2_1_1Value.html#a161550b3ef31b3a14b1d75149ba9ba71)());

const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv6 = xmpData["Xmp.dc.six"].value();
assert(getv6.toLong() == 0);
assert(getv6.ok());
assert(getv6.toFloat() == 0.0);
assert(getv6.ok());
assert(getv6.toRational() == Exiv2::Rational
   (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(0, 1));
assert(getv6.ok());

const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv7 = xmpData["Xmp.dc.seven"].value();
getv7.toLong(); // this should fail
assert(!getv7.ok());

const Exiv2::Value (doc/classExiv2_1_1Value.html) &getv8 = xmpData["Xmp.dc.eight"].value();
assert(getv8.toLong() == 1);
assert(getv8.ok());
assert(getv8.toFloat() == 1.0);
assert(getv8.ok());
assert(getv8.toRational() == Exiv2::Rational
   (doc/namespaceExiv2.html#a95756f3f7fa19103f83addf5fa088a30)(1, 1));
assert(getv8.ok());

// Deleting an XMP property
Exiv2::XmpData::iterator (doc/classExiv2_1_1XmpData.html#a6ad054efbea675843895e3f74c3c1923)
   pos = xmpData.findKey (doc/classExiv2_1_1XmpData.html#af4d4e63ed5641dbc6e211b880f6d0990)
   (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.dc.eight"));
if (pos == xmpData.end (doc/classExiv2_1_1XmpData.html#a1db4d5a92a7ec0694da08a7dee58faac)
   ()) throw Exiv2::Error (doc/classExiv2_1_1BasicError.html)(1, "Key not found");
xmpData.erase (doc/classExiv2_1_1XmpData.html#aa608042a71623e7dac640c135cb768e6)(pos);

// -----------------------------------------------------------------------
// Exiv2 has specialized values for simple XMP properties, arrays of simple
// properties and language alternatives.

// Add a simple XMP property in a known namespace
Exiv2::Value::AutoPtr (doc/classExiv2_1_1Value.html#a0f62e585b82c97738858b743e60dff21) v =
   Exiv2::Value::create (doc/classExiv2_1_1Value.html#ad6ff043921cd1a5c399a9a4fc8257006)
   (Exiv2::xmpText
   (doc/namespaceExiv2.html#a5153319711f35fe81cbc13f4b852450ca77cea60f60ef2c6f0f986137c5404
   c02));
v->read("image/jpeg");
xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
   (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.dc.format"), v.get());

// Add an ordered array of text values.
```

```
    v = Exiv2::Value::create (doc/classExiv2_1_1Value.html#ad6ff043921cd1a5c399a9a4fc8257006)
       (Exiv2::xmpSeq
       (doc/namespaceExiv2.html#a5153319711f35fe81cbc13f4b852450ca969c20e44455272599e3a27347154
       6e8)); // or xmpBag or xmpAlt.
    v->read("1) The first creator");         // The sequence in which the array
    v->read("2) The second creator");        // elements are added is their
    v->read("3) And another one");           // order in the array.
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.dc.creator"), v.get());

    // Add a language alternative property
    v = Exiv2::Value::create (doc/classExiv2_1_1Value.html#ad6ff043921cd1a5c399a9a4fc8257006)
       (Exiv2::langAlt
       (doc/namespaceExiv2.html#a5153319711f35fe81cbc13f4b852450ca52dce1d022dd8927bc651d2e51dc1
       bcd));
    v->read("lang=de-DE Hallo, Welt");       // The default doesn't need a
    v->read("Hello, World");                 // qualifier
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.dc.description"), v.get());

    // According to the XMP specification, Xmp.tiff.ImageDescription is an
    // alias for Xmp.dc.description. Exiv2 treats an alias just like any
    // other property and leaves it to the application to implement specific
    // behaviour if desired.
    xmpData["Xmp.tiff.ImageDescription"] = "TIFF image description";
    xmpData["Xmp.tiff.ImageDescription"] = "lang=de-DE TIFF Bildbeschreibung";

    // -------------------------------------------------------------------------
    // Register a namespace which Exiv2 doesn't know yet. This is only needed
    // when properties are added manually. If the XMP metadata is read from an
    // image, namespaces are decoded and registered at the same time.
    Exiv2::XmpProperties::registerNs
       (doc/classExiv2_1_1XmpProperties.html#ae58ee081625b7924563e93a1ba184fec)("myNamespace/",
       "ns");

    // -------------------------------------------------------------------------
    // Add a property in the new custom namespace.
    xmpData["Xmp.ns.myProperty"] = "myValue";

    // -------------------------------------------------------------------------
    // There are no specialized values for structures, qualifiers and nested
    // types. However, these can be added by using an XmpTextValue and a path as
    // the key.

    // Add a structure
    Exiv2::XmpTextValue (doc/classExiv2_1_1XmpTextValue.html) tv("16");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpDM.videoFrameSize/stDim:w"),
       &tv);
    tv.read("9");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpDM.videoFrameSize/stDim:h"),
       &tv);
    tv.read("inch");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpDM.videoFrameSize/stDim:unit"),
       &tv);

    // Add an element with a qualifier (using the namespace registered above)
    xmpData["Xmp.dc.publisher"] = "James Bond";  // creates an unordered array
    xmpData["Xmp.dc.publisher[1]/?ns:role"] = "secret agent";

    // Add a qualifier to an array element of Xmp.dc.creator (added above)
    tv.read("programmer");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.dc.creator[2]/?ns:role"), &tv);

    // Add an array of structures
    tv.read("");                                         // Clear the value
    tv.setXmpArrayType(Exiv2::XmpValue::xaBag);
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpBJ.JobRef"), &tv); // Set the
       array type.

    tv.setXmpArrayType(Exiv2::XmpValue::xaNone);
    tv.read("Birthday party");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpBJ.JobRef[1]/stJob:name"), &tv);
    tv.read("Photographer");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpBJ.JobRef[1]/stJob:role"), &tv);

    tv.read("Wedding ceremony");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpBJ.JobRef[2]/stJob:name"), &tv);
    tv.read("Best man");
    xmpData.add (doc/classExiv2_1_1XmpData.html#a8ce28ae5c68a30b8e646c7ddfed75843)
       (Exiv2::XmpKey (doc/classExiv2_1_1XmpKey.html)("Xmp.xmpBJ.JobRef[2]/stJob:role"), &tv);

    // Add a creator contact info structure
    xmpData["Xmp.iptc.CreatorContactInfo/Iptc4xmpCore:CiAdrCity"] = "Kuala Lumpur";
    xmpData["Xmp.iptc.CreatorContactInfo/Iptc4xmpCore:CiAdrCtry"] = "Malaysia";
    xmpData["Xmp.iptc.CreatorContactInfo/Iptc4xmpCore:CiUrlWork"] = "http://www.exiv2.org";

    // -----------------------------------------------------------------------
    // Output XMP properties
    for (Exiv2::XmpData::const_iterator
       (doc/classExiv2_1_1XmpData.html#a9c0a6575296f3da8bfc200091da40f2e) md = xmpData.begin
       (doc/classExiv2_1_1XmpData.html#aa6649bbd9d1f35555778febb49d5857a)();
         md != xmpData.end (doc/classExiv2_1_1XmpData.html#a1db4d5a92a7ec0694da08a7dee58faac)
       (); ++md) {
        std::cout << std::setfill(' ') << std::left
                  << std::setw(44)
                  << md->key() << " "
                  << std::setw(9) << std::setfill(' ') << std::left
                  << md->typeName() << " "
                  << std::dec << std::setw(3)
                  << std::setfill(' ') << std::right
                  << md->count
       (doc/classExiv2_1_1XmpData.html#a65b24c7bef3d7e9f2b58edfc19571753)() << "  "
                  << std::dec << md->value()
                  << std::endl;
    }

    // -----------------------------------------------------------------------
    // Serialize the XMP data and output the XMP packet
    std::string xmpPacket;
    if (0 != Exiv2::XmpParser::encode
       (doc/classExiv2_1_1XmpParser.html#afad88c80404f9f35b687b33fe9ea9c63)(xmpPacket,
       xmpData)) {
        throw Exiv2::Error (doc/namespaceExiv2.html#accd3e49cafe9db52c1e0e6f648753cae)(1,
       "Failed to serialize XMP data");
    }
    std::cout << xmpPacket << "\n";

    // Cleanup
    Exiv2::XmpParser::terminate
       (doc/classExiv2_1_1XmpParser.html#a46ff7c85b860ef81310e0ac8dd6b62a2)();

    return 0;
}
catch (Exiv2::AnyError (doc/classExiv2_1_1AnyError.html)& e) {
    std::cout << "Caught Exiv2 exception '" << e << "'\n";
    return -1;
}
```

The resulting XMP Exiv2 metadata and XMP packet is below. The same can be achieved with a set of commands (sample.html#xmp) to the Exiv2 command line tool.

```
$ xmpsample
Xmp.dc.source                            XmpText    13  xmpsample.cpp
Xmp.dc.subject                           XmpBag      2  Palmtree, Rubbertree
Xmp.dc.title                             LangAlt     2  lang="de-DE" Sonnenuntergang am Strand
Xmp.dc.one                               XmpText     2  -1
Xmp.dc.two                               XmpText     6  3.1415
Xmp.dc.three                             XmpText     3  5/7
Xmp.dc.four                              XmpText     3  255
Xmp.dc.five                              XmpText     3  256
Xmp.dc.six                               XmpText     5  false
Xmp.dc.seven                             XmpText     5  Seven
Xmp.dc.format                            XmpText    10  image/jpeg
Xmp.dc.creator                           XmpSeq      3  1) The first creator, 2) The second cr
Xmp.dc.description                       LangAlt     2  lang="x-default" Hello, World, lang="d
Xmp.tiff.ImageDescription                LangAlt     2  lang="x-default" TIFF image descriptio
Xmp.xmpDM.videoFrameSize/stDim:w         XmpText     2  16
Xmp.xmpDM.videoFrameSize/stDim:h         XmpText     1  9
Xmp.xmpDM.videoFrameSize/stDim:unit      XmpText     4  inch
Xmp.dc.publisher                         XmpBag      1  James Bond
Xmp.dc.publisher[1]/?ns:role             XmpText    12  secret agent
Xmp.dc.creator[2]/?ns:role               XmpText    10  programmer
Xmp.xmpBJ.JobRef                         XmpText     0  type="Bag"
Xmp.xmpBJ.JobRef[1]/stJob:name           XmpText    14  Birthday party
Xmp.xmpBJ.JobRef[1]/stJob:role           XmpText    12  Photographer
Xmp.xmpBJ.JobRef[2]/stJob:name           XmpText    16  Wedding ceremony
Xmp.xmpBJ.JobRef[2]/stJob:role           XmpText     8  Best man

<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 4.1.1-Exiv2">
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about=""
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:ns="myNamespace/"
    xmlns:tiff="http://ns.adobe.com/tiff/1.0/"
    xmlns:xmpDM="http://ns.adobe.com/xmp/1.0/DynamicMedia/"
    xmlns:stDim="http://ns.adobe.com/xap/1.0/sType/Dimensions#"
    xmlns:xapBJ="http://ns.adobe.com/xap/1.0/bj/"
    xmlns:stJob="http://ns.adobe.com/xap/1.0/sType/Job#"
   dc:source="xmpsample.cpp"
   dc:one="-1"
   dc:two="3.1415"
   dc:three="5/7"
   dc:four="255"
   dc:five="256"
   dc:six="false"
   dc:seven="Seven"
   dc:format="image/jpeg">
   <dc:subject>
    <rdf:Bag>
     <rdf:li>Palmtree</rdf:li>
     <rdf:li>Rubbertree</rdf:li>
    </rdf:Bag>
   </dc:subject>
   <dc:title>
    <rdf:Alt>
     <rdf:li xml:lang="de-DE">Sonnenuntergang am Strand</rdf:li>
     <rdf:li xml:lang="en-US">Sunset on the beach</rdf:li>
    </rdf:Alt>
   </dc:title>
   <dc:creator>
    <rdf:Seq>
     <rdf:li>1) The first creator</rdf:li>
     <rdf:li rdf:parseType="Resource">
      <rdf:value>2) The second creator</rdf:value>
      <ns:role>programmer</ns:role>
     </rdf:li>
     <rdf:li>3) And another one</rdf:li>
    </rdf:Seq>
   </dc:creator>
   <dc:description>
    <rdf:Alt>
     <rdf:li xml:lang="x-default">Hello, World</rdf:li>
     <rdf:li xml:lang="de-DE">Hallo, Welt</rdf:li>
    </rdf:Alt>
   </dc:description>
   <dc:publisher>
    <rdf:Bag>
     <rdf:li rdf:parseType="Resource">
      <rdf:value>James Bond</rdf:value>
      <ns:role>secret agent</ns:role>
     </rdf:li>
    </rdf:Bag>
   </dc:publisher>
   <tiff:ImageDescription>
    <rdf:Alt>
     <rdf:li xml:lang="x-default">TIFF image description</rdf:li>
     <rdf:li xml:lang="de-DE">TIFF Bildbeschreibung</rdf:li>
    </rdf:Alt>
   </tiff:ImageDescription>
   <xmpDM:videoFrameSize
    stDim:w="16"
    stDim:h="9"
    stDim:unit="inch"/>
   <xapBJ:JobRef>
    <rdf:Bag>
     <rdf:li
      stJob:name="Birthday party"
      stJob:role="Photographer"/>
     <rdf:li
      stJob:name="Wedding ceremony"
      stJob:role="Best man"/>
    </rdf:Bag>
   </xapBJ:JobRef>
  </rdf:Description>
```

```
 </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

Exiv2 v0.28.0

Last modified 2023-05-14 13:52 UTC