

An XML-based bird image cataloging system



John W. Shipman

2009-10-11 13:13

Table of Contents

1. Requirements	1
1.1. How to get this publication	2
2. Design notes	2
3. The schema	3
3.1. Schema prologue	3
3.2. <code>image-catalog</code> : The root element	3
3.3. <code>original</code> : Data for one exposure	3
3.4. The <code>info</code> pattern	5
4. Small example	6
5. <code>birdimages.py</code> : A Python object for the catalog	6
5.1. The <code>ImageCatalog</code> object	6
5.2. <code>class Original</code> : One bird slide catalog entry	7
6. <code>birdimages.py</code> : Internals	8
6.1. Prologue	8
6.2. Imports	8
6.3. Manifest constants	8
6.4. <code>class ImageCatalog</code> : Catalog object	9
6.5. <code>ImageCatalog.__init__()</code> : Constructor	10
6.6. <code>ImageCatalog.addOriginal()</code> : Add a new catalog entry	10
6.7. <code>ImageCatalog.getOriginal()</code> : Retrieve an original by catalog number	11
6.8. <code>ImageCatalog.genOriginals()</code> : Generate all catalog entries	11
6.9. <code>ImageCatalog.genAb6()</code> : Generate all entries with a given bird code	12
6.10. <code>ImageCatalog.readFile()</code> : Read an XML file	12
6.11. <code>class Original</code> : One catalog entry	13
6.12. <code>Original.__init__()</code> : Constructor	14
6.13. <code>Original.readNode()</code> : Build a catalog entry from an <code>Element</code> node	14
6.14. <code>childText()</code> : Get text from a child node	15
6.15. <code>nodeText()</code> : Get text from an element	16
6.16. <code>cattest</code> : A test driver for <code>ImageCatalog</code>	17

1. Requirements

My collection of bird images runs into the thousands. I'd like to be able to manage it and be able to retrieve images by bird type, locality, time period, and such constraints.

Scanning an image in a proper scanner takes several minutes. I want to avoid wasting time scanning poor photos of species for which I already have better shots.

Because the days of film photography are drawing to a close, this system must also anticipate the transition to digital media.

1.1. How to get this publication

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to john@nmt.edu.

2. Design notes

The heart of an image management system is the *catalog number*, a unique identifier for original images. For many years preceding the design of this system, my catalog number had this format:

```
YYYYMMDDxFF
```

YYYYMMDD

The date as a four-digit year, two-digit month, and two-digit day. Note that either the *MM* part or the *DD* part may be “00” if the month or day is unknown.

x

In most cases, this separator will be a period (“.”). However, to address the problem of multiple shots on the same day with the same frame number, it may also be a lowercase letter.

While indexing older slides, my method was to use a period for the first or only slide with that frame number on that day, then assign letters a, b, ... as duplicates appeared for cataloging.

However, recently a better practice emerged: use a period here for the first or only roll of film (or digital camera) used on that day, then assign letters for additional rolls or cameras. This makes it easier to see the actual shooting sequence on a given day.

FF

For film originals, this is the frame number. For unmounted film, this number appears in the margin of the original; if the numbers 1, 2, ... appear *exactly* on the frame boundaries, the alternate sequence 1A, 2A, ... is used, but the trailing A is omitted from the image's catalog number.

For most all the older mounted 35mm slides, this catalog number appears in ink on the actual slide mount. Holding the slide in its correct orientation, rotate it around the horizontal axis, and the catalog number is written on what is now the top edge. Typically the word “Kodachrome” (or whatever) will appear right-side-up just below this edge.

Digital cameras typically assign a four-digit number to each frame. My Canon A530, for example, names each image “IMG_ *nnnn* .JPG”, where *nnnn* starts at 0001 and eventually wraps around. Assuming that one never shoots more than 10,000 frames a day, this number plus the date should uniquely identify a frame for a given camera.

I also try to avoid long, featureless strings of digits nowadays, as they are hard to read. The Tech Computer Center's ticket system, for example, assigns each ticket a number of the form *YYYYMMDDh-hmmss*, but the resulting lump of 14 featureless digits can be hard to parse, especially in a small screen font. There is an ISO date standard that recommends dashes as separators, so a full timestamp would look like “1949-09-05T14:30:15”.

Here, then, is the current general form of the catalog number for originals:

¹ <http://www.nmt.edu/~john/scans/slides/ims/>

² <http://www.nmt.edu/~john/scans/slides/ims/slidecat.pdf>

```
YYYY-MM-DDxFFFF
```

where the normal character for *x* is a hyphen (“-”) instead of a period. The use of lowercase letters for the *x* separator is still supported, for disambiguating film frames with the same number on the same date, or for disambiguating frames from different cameras with the same number.

3. The schema

I’ve been pretty happy with XML as a general tool for managing complex information, especially since the advent of the Relax NG compact format (RNC)³ schema language and the `nxml-emacs`⁴ XML-aware text editor that make it so much easier to create and maintain XML files.

Below, in literate programming form⁵, is the schema that defines the slide catalog document type.

3.1. Schema prologue

Here is the opening comment block for the RNC schema.

birdimages.rnc

```
# slidecat.rnc: Relax NG schema for bird slide
# Documentation: http://www.nmt.edu/~john/scans/slides/
#-----
```

3.2. image-catalog: The root element

The root element for the entire XML document type is `image-catalog`:

birdimages.rnc

```
start = image-catalog

image-catalog = element image-catalog
{ attribute date { text }?,      1
  attribute revision { text }?,  2
  original*                      3
}
```

- 1** The `date` attribute is intended for an RCS Date tag or other content indicating when the file was last touched.
- 2** Intended for an RCS Revision tag or the equivalent.
- 3** Each primary entry in the catalog is an `original` element.

3.3. original: Data for one exposure

Each `original` element describes one exposure, whether on film or digital media.

birdimages.rnc

```
original = element original
{ attribute cat-no { cat-no-pattern },  1
```

³ <http://www.nmt.edu/tcc/help/pubs/rnc/>

⁴ <http://www.nmt.edu/tcc/help/pubs/nxml/>

⁵ <http://www.nmt.edu/tcc/help/lang/python/examples/litsource>

```

attribute ab6 { text },           2
attribute state                   3
{ xsd:string { pattern='[a-z]{2,3}' }},
attribute qual                   4
{ xsd:string { pattern='[abcdf]' } }?,
attribute scan { xsd:positiveInteger }?, 5
attribute old { old-pattern }?, 6
attribute arch-no { text }?,    7
info                             8
}

```

The `cat-no-pattern` pattern defines the format of a catalog number.

birdimages.rnc

```

cat-no-pattern = xsd:string
{ pattern='[0-9]{4}-[0-9]{2}-[0-9]{2}[\-a-z][0-9]{4}'
}

```

The `old-pattern` pattern defines the format of an old-style catalog number.

birdimages.rnc

```

old-pattern = xsd:string
{ pattern='[0-9]+\.[0-9]{2}'
}

```

- 1 The catalog number is required and it must be unique within the file.
- 2 A list of one or more six-letter bird codes representing species in the image; each code may be followed by a ? character if the identity is not certain. For the bird code system, see *A system for representing taxonomic nomenclature*⁶. Examples:

```

ab6='amerob'
ab6='larbun?'
ab6='eargre amecoo'
ab6='dowwoo|haiwoo'
ab6='amewig^eurwig?'

```

- 3 U.S. postal code or the foreign equivalent. Two or three lowercase letters.
- 4 Optional quality indicator. Quality a is reserved for really stunning photography, suitable for contests or commercial sale; b is a pretty good picture; c is decent; d is poor (the default); and quality code f is reserved only for really horrible pictures that are in the catalog only because they are the only documentation there is.
- 6 Catalog number under the old system. Originally I assigned serial numbers to each roll, so for example "#59.38" would mean roll 58, frame 38. There are a few undated originals with these notations, so I can sometimes use the old catalog numbers to approximate their dates.
- 7 This attribute is obsolete. Originally I was going to record in the image catalog which archive directory contained the reference image. Later I decided to put this information somewhere else; see *archx: A program to index a photo archive*⁷.

Formerly, if this image were contained in one of the standard PNG archive CDs, this attribute is the three-digit archive number. For example, `arch-no='003'` means the image was contained on the PNG-003 CD. This has been abandoned; bird images are now in `www/scans/bird/bird-NNN`.

⁶ <http://www.nmt.edu/~shipman/z/cbc/nomo.html>

⁷ <http://www.nmt.edu/~john/scans/slides/ims>

- 5 If the original is film, this attribute specifies the dot pitch, in dots per inch, of the scanner used to produce the digital image. For example, for a scanner that produces 4000dpi, the attribute would be `scan='4000'`. Omitted for digital originals.
- 8 The children of this element can be any of the elements in the `info` pattern.

3.4. The `info` pattern

Several elements describing the image can occur as children of the `original` element.

birdimages.rnc

```

info = ( loc? & note? & film? & light? & beh? & desc? & pose? )
film  = element film
{ attribute iso { iso-pattern }?,
  text
}
iso-pattern = xsd:positiveInteger
loc      = element loc { text }
note    = element note { text }
light   = element light { text }
beh     = element beh { text }
desc    = element desc { text }
pose    = element pose { text }

```

The content of the `film` element may be either a filmstock description such as “KR” for Kodachrome, or a digital camera type.

Each of the remaining choices is just a container for text. Their meanings are:

film

Filmstock or digital camera type. Common codes include KR for Kodachrome 64; KL for Kodachrome 200; EL for Ektachrome 400; Fj for Fuji Reala; and VC for Kodak VC-400.

The optional `iso` attribute is the ASA or ISO sensitivity rating.

loc

Description of the locality. Use colons to separate levels, e.g., “<loc>Bosque del Apache: Headquarters</loc>”.

note

General notes about the image that don't fit the other categories. There can be multiple `note` elements; each will be treated as a separate paragraph.

light

Comments on the lighting. Backlit, sidelit, direct (from behind the camera); strong or weak light; filtered; polluted; low-angle; and so on.

beh

Notes on the bird's behavior, e.g., flight, squawk (mouth is open).

desc

Plumage or other visual description of the bird.

pose

How the bird is posed: frontal, front quarter, profile, rear quarter, butt-shot.

4. Small example

Here is a small fragment containing two complete catalog entries:

```
<?xml version='1.0' encoding='UTF-8'?>
<image-catalog>
  <original state='nm' ab6='brespa' cat-no='1980-12-29-0037'>
    <loc>Sedillo Spring</loc>
    <note>sm, profile</note>
  </original>
  <original state='nm' ab6='uplsan' cat-no='2003-09-06-0017'>
    <loc>Artesia</loc>
    <note>rear qtr, back sharp, head soft</note>
    <film>Fujichrome 400</film>
  </original>
</image-catalog>
```

5. `birdimages.py`: A Python object for the catalog

The `birdimages.py` module is a Python module containing a class that represents the contents of a slide catalog.

This section describes the external interface to the module. For the actual code of the module, see Section 6, “`birdimages.py`: Internals” (p. 8).

5.1. The `ImageCatalog` object

To access the image catalog:

```
from birdimages.py import *
C = ImageCatalog.readFile(filename)
```

where *filename* is the name of the image catalog. If omitted, this file name defaults to “`birdimages.xml`” in the current directory. If the file does not exist or is not well-formed, this method will raise `IOError`.

The code above would set `cat` to an instance of the `ImageCatalog` class. An `ImageCatalog` instance `C` exports these methods:

`C.getOriginal(catNo)`

If the catalog contains an original image whose catalog number matches `catNo`, the method returns an `Original` instance that represents the catalog entry. See Section 5.2, “class `Original`: One bird slide catalog entry” (p. 7).

If the catalog contains no image with that number, the method raises a `KeyError` exception.

`C.genOriginals()`

Using a Python generator, this method generates all the entries in `C` as instances of class `Original`, in ascending order by catalog number.

C.genAb6(birdId)

This method generates, as a sorted sequence of `Original` instances, all the catalog entries that contain a given kind of bird. The argument `birdId` is a string whose syntax is described in *A system for representing bird taxonomy*⁸, in the section on the `abbr.py` module.

Typically the argument is a six-letter bird code (e.g., "BALEAG" for Bald Eagle), but it may also be a string representing a species pair (e.g., "DOWWOO|HAIWOO" for Downy or Hairy Woodpecker) or a hybrid (e.g., "AMEWIG^EURWIG" for American × European Wigeon).

A question mark may be appended to any of the above values, e.g., "HAMFLY?" or "AMEWIG^EURWIG?". If a form appears both with and without the question mark, that is considered two different forms.

5.2. class Original: One bird slide catalog entry

An instance of this class represents one catalog entry. The related schema pattern is described in Section 3.3, "original: Data for one exposure" (p. 3).

An instance `O` of class `Original` exports these read-only attributes:

O.catNo

The catalog number.

O.ab6

The full `ab6` attribute, containing one or more bird ID strings.

O.state

State code.

O.qual

The `qual` attribute, if any, otherwise an empty string.

O.scan

The `scan` attribute as an integer, if present; otherwise `None`.

O.loc

The contents of the `loc` element, or an empty string if that element is missing.

O.note

The contents of the `note` element, or an empty string.

O.film

The text from the `film` element if present, or an empty string.

O.light

The light description from the `light` element, or an empty string.

O.beh

Behavior notes from the `beh` element, or an empty string.

O.desc

Description from the `desc` element, or an empty string.

O.pose

Posing information from the `pose` element, or an empty string.

⁸ <http://www.nmt.edu/~shipman/xnomo/>

6. `birdimages.py`: Internals

This section contains the actual code of the `birdimages.py` module in lightweight literate form. For more information on this methodology, see the author's [Lightweight Literate Programming](#) page⁹.

6.1. Prologue

The `birdimages.py` module starts with a brief module comment that points back to this documentation.

```
birdimages.py
"""birdimages.py: Python object for XML files using birdimages.rnc
    For documentation, see:
    http://www.nmt.edu/~john/www/scans/slides/ims/
    """
```

6.2. Imports

As always, we need the `sys` module for access to standard I/O streams.

```
birdimages.py
#=====
# Imports
#-----
import sys
```

To process the XML input file, we use the technique described in *Python XML processing with lxml*¹⁰. We'll use the name `et` for that implementation of the `ElementTree` interface.

```
birdimages.py
from lxml import etree as et
```

We'll also need global declarations for all the XML element and attribute names from our RNC schema. Rather than attempt to maintain these declarations in parallel with the schema itself, we use the tool described in *pyrang: A single-sourcing tool for Python-XML applications*¹¹. This program reads the Relax NG version of the schema file and writes a module named `rnc_slidecat.py` containing Python statements that set up the value of these variables.

The variables generated by *pyrang* have this general form:

```
RNC_name_suffix
```

where the *name* is the element or attribute name, and the *suffix* is "N" for element names and "A" for attribute names. For example, the variable for the `original` element is `"RNC_ORIGINAL_N"`.

```
birdimages.py
from rnc_slidecat import *
```

6.3. Manifest constants

There is one global constant (other than the ones imported from `rnc_slidecat.py`): the default image catalog name.

⁹ <http://www.nmt.edu/~shipman/soft/litprog/>

¹⁰ <http://www.nmt.edu/tcc/help/pubs/pylxml/>

¹¹ <http://www.nmt.edu/tcc/help/lang/python/examples/pyrang/>

```
#=====
# Manifest constants
#-----

DEFAULT_FILENAME = "birdimages.xml"
```

6.4. class ImageCatalog: Catalog object

An instance of this class represents the entire catalog file. The constructor is not intended for direct instantiation, and returns only an empty catalog. The static method `ImageCatalog.readFile()` does all the work of filling the empty catalog object from the XML input.

Here is the class declaration and external interface.

```
# - - - - - c l a s s   I m a g e C a t a l o g   - - - - -

class ImageCatalog:
    """Represents the entire catalog.

    Exports:
    ImageCatalog():
        [ returns a new, empty ImageCatalog object ]
    .addOriginal(o):
        [ o is an Original object ->
          if self contains an original with the same catalog number
          as o ->
            raise KeyError
          else ->
            self := self with o added ]
    .getOriginal(catNo):
        [ catNo is a catalog number as a string ->
          if self has an original whose catalog number
          matches catNo ->
            return that original as an Original object
          else -> raise KeyError ]
    .genOriginals():
        [ generate the Originals in self in catalog number order ]
    .genAb6(code):
        [ code is a birdId string ->
          generate the Originals in self whose .ab6
          attributes contain code ]
    ImageCatalog.readFile(f):    # Static method
        [ f names a readable file valid against birdimages.rnc,
          defaulting to DEFAULT_FILENAME ->
          return a new ImageCatalog representing that file ]
```

Here are the class's internal state items.

State/Invariants:

```

    .__catNoMap:
        [ a dictionary whose values are the Originals in self,
          and each key is the value's .catNo ]
    .__ab6Map:
        [ a dictionary whose keys are all the birdId strings
          that appear in self's .ab6 attributes (uppercased and
          right-blank-padded to full length), and each
          corresponding value is a list of Originals
          that contain that key in their .ab6 attributes ]
    """

```

The `.__ab6Map` dictionary exists to support the `.genAb6()` method. Note that a given `Original` can appear in more than one of the lists that are values of the `.__ab6Map` dictionary. For example, an original with the XML attribute `"ab6="virrai sora"` would appear in the lists for both `.__ab6Map["VIRRAI"]` and `.__ab6Map["SORA "]`.

6.5. ImageCatalog.__init__(): Constructor

This trivial constructor simply creates the two internal dictionaries, initially empty.

birdimages.py

```

# - - -   I m a g e C a t a l o g . _ _ i n i t _ _   - - -

def __init__ ( self ):
    """Constructor for ImageCatalog.
    """
    self.__catNoMap = {}
    self.__ab6Map   = {}

```

6.6. ImageCatalog.addOriginal(): Add a new catalog entry

This method takes an `Original` instance and stores it in `self`.

birdimages.py

```

# - - -   I m a g e C a t a l o g . a d d O r i g i n a l   - - -

def addOriginal ( self, o ):
    """Add an original to the catalog.
    """

```

First we add the new entry to the `.__catNoMap` dictionary. Duplicates are not allowed, so we check to insure there wasn't already an entry for that catalog number.

birdimages.py

```

#-- 1 --
# [ if self.__catNoMap has an entry for o.catNo ->
#     raise KeyError
#     else ->
#     self.__catNoMap[o.catNo] = o ]
if self.__catNoMap.has_key(o.catNo):
    raise KeyError, "Duplicate catalog number '%s'" % o.catNo
self.__catNoMap[o.catNo] = o

```

Adding the new entry to the `.__ab6Map` dictionary is a bit more complicated. Because the XML `ab6` attribute can have multiple codes separated by spaces, we must use the Python `.split()` function to

get a set of code strings. For example, if the original attribute is "buwtea^cintea norsho?", that will be indexed on two strings, "buwtea^cintea" and "norsho?".

Then, the first time we observe a code, we set up the dictionary value with a new list containing the `Original`, but once we've seen that code, we append the code to the list.

birdimages.py

```
#-- 2 --
# [ self.__ab6Map += entries mapping code |-> o
#     for all codes in o.ab6 ]
codeList = o.ab6.split()
for code in codeList:
    key = code.rstrip().upper()
    try:
        self.__ab6Map[key].append(o)
    except KeyError:
        self.__ab6Map[key] = [o]
```

6.7. `ImageCatalog.getOriginal()`: Retrieve an original by catalog number

This method uses the `.__catNoMap` dictionary to look up the original by catalog number. It raises `KeyError` if the dictionary does not have that key value.

birdimages.py

```
# - - -   I m a g e C a t a l o g . g e t O r i g i n a l   - - -

def getOriginal ( self, catNo ):
    """Retrieve the original with a given catalog number.
    """
    return self.__catNoMap[catNo]
```

6.8. `ImageCatalog.genOriginals()`: Generate all catalog entries

This method first extracts a list of all the keys in the `.__catNoMap` dictionary, then sorts them, then generates the values using that sorted list.

birdimages.py

```
# - - -   I m a g e C a t a l o g . g e n O r i g i n a l s   - - -

def genOriginals ( self ):
    """Generate all originals in catalog order"""
    keyList = self.__catNoMap.keys()
    keyList.sort()
    for key in keyList:
        yield self.__catNoMap[key]
    raise StopIteration
```

6.9. ImageCatalog.genAb6(): Generate all entries with a given bird code

If the catalog has any entries for a given code, that code will be a key in the `.__ab6Map` dictionary, and the corresponding value will be a list containing the matching `Original` instances, which we then generate. If there is no such key, we raise `KeyError`.

birdimages.py

```
# - - -   I m a g e C a t a l o g . g e n A b 6   - - -

def genAb6 ( self, code ):
    """Retrieve originals with a given bird code.
    """

    #-- 1 --
    # [ if self.__ab6Map has a key that matches code ->
    #     resultList := the corresponding value
    #     else -> raise KeyError ]
    resultList = self.__ab6Map[code.rstrip().upper()]

    #-- 2 --
    # [ generate the elements of resultList ]
    for result in resultList:
        yield result
    raise StopIteration
```

6.10. ImageCatalog.readFile(): Read an XML file

This static method takes a file name as an argument and, assuming the file is well-formed, builds an `et.ElementTree` instance representing the file. It then walks the tree, converting each `original` element to a catalog entry and adding it to `self`.

birdimages.py

```
# - - -   I m a g e C a t a l o g . r e a d F i l e   - - -   S t a t i c

def readFile ( fileName=DEFAULT_FILENAME ):
    """Read an XML file, return it as an ImageCatalog.
    """

    #-- 1 --
    # [ if fileName is a readable, well-formed XML file ->
    #     doc := an et.ElementTree instance representing the file
    #     else -> raise IOError ]
    try:
        doc = et.parse ( fileName )
    except IOError, detail:
        raise IOError, ( "Can't read the catalog file '%s': %s" %
            (fileName, detail) )
    except et.XMLSyntaxError, detail:
        raise IOError, ( "Catalog file '%s' not well-formed: %s" %
            (fileName, detail) )
```

First we instantiate a new, empty `ImageCatalog` object to which we can add the entries from the tree.

```
#-- 2 --
# [ cat := a new, empty ImageCatalog object ]
cat = ImageCatalog()
```

To get all the RNC_ORIGINAL_N children of doc, we'll use the .getiterator() function.

```
#-- 3 --
# [ cat := cat with Original instances added, made from
#       the RNC_ORIGINAL_N children of doc ]
for oNode in doc.getiterator ( RNC_ORIGINAL_N ):
```

For the logic that converts the XML representation of each catalog entry into an Original object, see Section 6.13, "Original.readNode(): Build a catalog entry from an Element node" (p. 14).

```
#-- 3 loop --
# [ oNode is an RNC_ORIGINAL_N node ->
#     result := result with a new original added
#           made from oNode ]
cat.addOriginal ( Original.readNode ( oNode ) )

#-- 4 --
return cat

readFile = staticmethod ( readFile )
```

6.11. class Original: One catalog entry

Each instance of this class represents one XML original element. Here is the class interface:

```
# - - - - - class Original - - - - -

class Original:
    """Represents one image catalog entry.

    Exports:
    Original ( catNo, ab6, state, qual='', loc='', note='',
              film='', light='', beh='', desc='', pose='' ):
    [ (catNo is the catalog number as a string) and
      (ab6 is a space-separated list of bird-ID strings) and

      (state is a two-letter US postal code) and
      (qual is a quality rating or '') and
      (loc is locality text or '') and
      (note is note text or '') and
      (film contains filmstock comments or '') and
      (light contains lighting comments or '') and
      (beh contains behavior comments or '') and
      (desc contains plumage details or '') and
      (pose contains pose comments or '') ->
      return a new Original object containing those
```

```

        values ]
    .catNo:      [ as passed to constructor, read-only ]
    .ab6:       [ as passed to constructor, read-only ]
    .state:     [ as passed to constructor, read-only ]
    .qual:      [ as passed to constructor, read-only ]
    .loc:       [ as passed to constructor, read-only ]
    .note:      [ as passed to constructor, read-only ]
    .film:      [ as passed to constructor, read-only ]
    .light:     [ as passed to constructor, read-only ]
    .beh:       [ as passed to constructor, read-only ]
    .desc:      [ as passed to constructor, read-only ]
    .pose:      [ as passed to constructor, read-only ]
Original.readNode(node): # Static method
    [ node is an RNC_ORIGINAL_N et.Element ->
      if node is valid against birdimages.rnc ->
        return a new Original object representing that
        element
      else -> raise IOError ]
"""

```

6.12. Original.__init__(): Constructor

This straightforward constructor just stores all the argument values in the instance.

birdimages.py

```

# - - -   O r i g i n a l . _ _ i n i t _ _   - - -

def __init__( self, catNo, ab6, state, qual='', scan='',
              loc='', note='', film='', light='', beh='', desc='', pose='' ):
    """Constructor for Original.
    """
    self.catNo = catNo
    self.ab6   = ab6
    self.state = state
    self.qual  = qual
    self.scan  = scan
    self.loc   = loc
    self.note  = note
    self.film  = film
    self.light = light
    self.beh   = beh
    self.desc  = desc
    self.pose  = pose

```

6.13. Original.readNode(): Build a catalog entry from an Element node

This static method operates on an et.Element instance that represents an original element. Assuming that it is valid, it returns a new Original instance representing that element.

```
# - - -   O r i g i n a l . r e a d N o d e   - - -   S t a t i c   m e t h o d

def readNode ( node ):
    """Translate an original element into an Original object.
    """
```

We could do a lot of error checking here, and our intended function entitles us to throw an `IOError` exception if the file isn't valid. However, at the moment I prepare the files using *nxml-emacs*, which continuously validates the file. This allows us to assume here that everything is valid.

Because an `et.Element`'s `.attrib` attribute works like a dictionary, we can use the usual dictionary `.get()` method to supply default values for missing attributes.

```
#-- 1 --
catNo = node.attrib.get ( RNC_CAT_NO_A, None )
ab6   = node.attrib.get ( RNC_AB6_A, None )
state = node.attrib.get ( RNC_STATE_A, None )
qual  = node.attrib.get ( RNC_QUAL_A, None )
rawScan = node.attrib.get ( RNC_SCAN_A, None )

if rawScan: scan = int ( rawScan )
else:       scan = None
```

For the values that live in child nodes, we use Section 6.14, “`childText(): Get text from a child node`” (p. 15).

```
#-- 2 --
loc   = childText ( node, RNC_LOC_N )
note  = childText ( node, RNC_NOTE_N )
film  = childText ( node, RNC_FILM_N )
light = childText ( node, RNC_LIGHT_N )
beh   = childText ( node, RNC_BEH_N )
desc  = childText ( node, RNC_DESC_N )
pose  = childText ( node, RNC_POSE_N )

#-- 3 --
return Original ( catNo, ab6, state, qual, scan, loc, note,
                  film, light, beh, desc, pose )

readNode = staticmethod ( readNode )
```

6.14. `childText(): Get text from a child node`

This utility function looks for a child node with a given name and, if one is found, returns all the text nodes in and under that child node. If there is no child by that name, it returns an empty string.

```
# - - -   c h i l d T e x t   - - -

def childText ( node, childName ):
    """Return the textual content of a child node, if any.
```

```

    [ (node is an et.Element) and
      (childName is a string) ->
        if node has any child nodes named childName ->
          return a Unicode string containing the concatenation
            of all text node descendants of those children
        else -> return '' ]
    "" ""

```

First we use an XPath expression to get a list of the matching child nodes.

birdimages.py

```

#-- 1 --
# [ node is an et.Element ->
#   childList := a list of all children of node named childName ]
childList = node.xpath ( childName )

```

Whether this list is empty or not, we then create a list containing the text from each entry. Then we concatenate the elements of that list and return that as a result.

birdimages.py

```

#-- 2 --
# [ childList is a node-set ->
#   textList := a list of the text descendants from each
#             node in childList ]
textList = [ nodeText(c) for c in childList ]

#-- 3 --
return "".join ( textList )

```

6.15. nodeText (): Get text from an element

This helper function takes as an argument an `et.Element` instance, and returns a Unicode string containing the concatenation of the text content of that node and all its descendants.

birdimages.py

```

# - - -   n o d e T e x t   - - -

def nodeText ( node ):
    '''Returns text in and under an et.Element, as Unicode.

    [ node is an et.Element ->
      return the concatenation of all descendant Text nodes
        of node ]
    ...

```

First, we use an XPath expression to find all the text nodes under the given `node`. My first attempt at an XPath expression was `"text()"`, but that returns only the immediate text children of a node. Adding the axis specifier `"descendant-or-self::"` applies the `text()` function to the node and all its descendants. Finally, we concatenate the strings.

birdimages.py

```

#-- 1 --
# [ textList := a list of all text descendants of
#             node, in document order ]
return ''.join ( node.xpath ( 'descendant-or-self::text()' ) )

```

6.16. cattest: A test driver for ImageCatalog

This small script instantiates an ImageCatalog object and does these tests:

- Use `.getOriginal()` to retrieve an image we know to be in there (2005-09-05-0003).
- Use `.genAb6()` to retrieve all images of Yellow Warbler (`yelwar`), a modest number.
- Dump the entire catalog using `.genOriginals()`.

The script starts with the usual Unix script prologue line and our imports.

```
#!/usr/bin/env python
#=====
# cattest: Test the ImageCatalog object.
#   For documentation, see:
#   http://www.nmt.edu/~john/www/scans/slides/ims/
#-----

from birdimages import *
```

cattest

Next comes the main.

```
# - - -   m a i n   - - -

def main():
    """Main test driver.
    """
    cat = ImageCatalog.readFile()

    print "=== Test: .getOriginal('2005-09-05-0003')"
    orig = cat.getOriginal ( '2005-09-05-0003' )
    showOrig ( orig )

    print "\n\n=== Test: genAb6('yelwar')"
    warblerList = [x for x in cat.genAb6('yelwar')]
    for warbler in warblerList:
        showOrig ( warbler )

    print "\n\n=== Test: Generate all"
    for o in cat.genOriginals():
        showOrig ( o )
```

cattest

The `showOrig()` function displays all the components of an `Original`.

```
# - - -   s h o w O r i g   - - -

def showOrig ( orig ):
    """Display the contents of an Original object.
    """
    print ( "\n#%s (%s) %s: %s" %
            (orig.catNo, orig.ab6, orig.state, orig.loc ) ),
    if orig.qual: print "  qual=%s" % orig.qual,
    if orig.note: print "  note=%s" % orig.note,
```

cattest

```
if orig.film: print " film=%s" % orig.film,
if orig.light: print " light=%s" % orig.light,
if orig.beh: print " beh=%s" % orig.beh,
if orig.desc: print " desc=%s" % orig.desc,
if orig.pose: print " pose=%s" % orig.pose,
print
```

Finally, the epilogue, calling the `main()` defined earlier.

cattest

```
#=====
# Epilogue
#-----

if __name__ == "__main__":
    main()
```