

catweb: A Web-based bird photo catalog



John W. Shipman

2009-10-11 14:46

Abstract

Describes a program that indexes bird photos using Web pages.

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

Table of Contents

1. Introduction	2
2. Definitions	3
2.1. Ancestor	3
2.2. Contained in	3
2.3. Descendant	3
2.4. Referred to	3
2.5. Inverted English name	3
3. Operation of the <i>catweb</i> script	4
3.1. The index page	4
3.2. The form page	5
4. XHTML output	6
4.1. XHTML design considerations	6
4.2. XHTML for the index page	6
4.3. XHTML for the form page	7
5. Styling with CSS	9
6. Analysis notes	11
7. Design notes	12
8. Code prologue	13
8.1. Module imports	13
9. Manifest constants	14
9.1. CSS_STYLESHEET: Stylesheet name	14
9.2. FORM_SUBDIR: Form subdirectory name	14
9.3. FULL_FRAME_MM2: Full frame size for 35mm	14
9.4. INDEX_PAGE: Name of the index page	15
9.5. MM_PER_IN: Conversion factor	15
10. The main program	15
10.1. <code>addArchive()</code> : Process one archive's worth of catalog entries	16
10.2. <code>addArchImage()</code> : Add all forms for one image	17
10.3. <code>buildWeb()</code> : Build all Web pages	18
10.4. <code>webPage()</code> : Create an empty XHTML page	19

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

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

11. class TaxonPhotoSet: The taxonomic tree node	20
11.1. TaxonPhotoSet.__init__(): Constructor	22
11.2. TaxonPhotoSet.addArchImage(): Add one photo	22
11.3. TaxonPhotoSet.__addReferred(): Basis case	23
11.4. TaxonPhotoSet.__addContained(): Recursive case	24
11.5. TaxonPhotoSet.walk(): Walk the tree	25
11.6. TaxonPhotoSet.nForms(): Number of contained forms	26
11.7. TaxonPhotoSet.genForms(): Generate referred FormPhotoSet objects	26
11.8. TaxonPhotoSet.buildPage(): Construct XHTML	27
11.9. TaxonPhotoSet.__buildSingleForm(): The taxon has only one form	28
11.10. TaxonPhotoSet.__webName(): Fill in scientific and English names	29
11.11. TaxonPhotoSet.__buildMultiForms(): The taxon has multiple forms	31
11.12. TaxonPhotoSet.__buildFormLine(): Build one form link and form page	32
12. class FormPhotoSet: All photos for one name	33
12.1. FormPhotoSet.__init__(): Constructor	34
12.2. FormPhotoSet.addArchImage(): Add one photo	34
12.3. FormPhotoSet.genArchImages(): Generate contained photos	34
12.4. FormPhotoSet.pathName(): Form page relative path name	35
12.5. FormPhotoSet.buildPage(): Build the XHTML page for one form	36
12.6. formSet.__buildTable(): Build the image table	37
12.7. formSet.__addRow(): Generate one row of the table	38
12.8. FormPhotoSet.__addThumbnail(): Add the thumbnail cell	38
12.9. FormPhotoSet.__addSize(): Add the size cell	39
12.10. FormPhotoSet.__addData(): Add the cataloging data cell	40
13. Epilogue	41
14. Defect statistics	42
14.1. Syntax errors	42
14.2. Strong typing errors	42
14.3. Logic errors	44

1. Introduction

This document describes a program that builds a set of Web pages that index a collection of bird photos. See these related documents for other pieces of this system:

- *An XML-based image cataloging system*³. Start here with an explanation of the basic requirements for the system, and the format of the XML catalog of images.
- *archx: A program to index a photo archive*⁴. This document explains how scans are organized into CD-sized directories, and provides a program to build XML-based indices of the contents of each directory. It also reduces each image to a standard thumbnail size and places the thumbnails in a separate directory.
- *A system for representing bird taxonomy*⁵. Because it is customary to organize bird records in phylogenetic order, this system slots birds into biological classifications.

Once images are cataloged, scanned, and placed into archive directories, the *catweb* script builds a set of web pages that list all images according to what kinds of birds are found in each image, with thumbnail images.

³ <http://www.nmt.edu/~john/scans/slides/ims/>

⁴ <http://www.nmt.edu/~john/scans/slides/archx/>

⁵ <http://www.nmt.edu/~shipman/xnomo/>

2. Definitions

Here are some terms used throughout this document.

2.1. Ancestor

The term *ancestor* is used here in the sense of a tree structure. The ancestors of a node include its parent node (if any), its parent's parent, its parent's parent's parent, and so on all the way to the root of a tree.

In reference to the tree of taxa that represent a taxonomic arrangement, the ancestors of a taxon are the higher-level taxa that contain that taxon.

2.2. Contained in

A taxon is said to be *contained in* all its ancestor taxa (see Section 2.1, "Ancestor" (p. 3). For example, the species *Falco sparverius* is contained in genus *Falco*, subfamily Falconinae, family Falconidae, order Falconiformes, and class Aves.

2.3. Descendant

The term *descendant* is used in the sense of a tree structure: the descendants of a node in a tree are the children of the given node, the children of the children, and so forth down to the leaf nodes of each branch.

2.4. Referred to

Each English name describing a form of bird is *referred to* the one specific taxon that is the smallest unit of classification to which the image clearly refers. In the great majority of cases, a photo is referred to a species: for example, the Crissal Thrasher is referred to *Toxostoma crissale*.

However, there are both finer and coarser distinctions. Blue Goose is a color morph of the species Snow Goose; the base taxonomy system calls such groups "subspecific forms," meaning that they are finer than species level; technically, "subspecies" has a narrower meaning in taxonomy.

There are many cases where the image is referred to a higher-ranked taxon than species. For example, "owl sp." implies no finer level of detail than Order Strigiformes, the order of owls. Also, Consider hybrids such as Mallard (*Anas platyrhynchos*) × Gadwall (*Anas strepera*). This form is referred to genus *Anas*.

2.5. Inverted English name

The inverted form of an English bird name means that the generic part comes first, followed by a comma and the specific part. For example, "American Robin" becomes "Robin, American."

For this application, we define the generic part of a name as the last word or hyphenated group of words. If there is only one name or hyphenated group, the inverted name is the same as the uninverted name. Examples:

Name	Inverted name
Aplomado Falcon	Falcon, Aplomado
Fork-tailed Storm-Petrel	Storm-Petrel, Fork-tailed
Sora	Sora

Name	Inverted name
Whip-poor-will	Whip-poor-will

3. Operation of the *catweb* script

The script is intended to be run in the base directory above the `indices/`, `thumb/`, and archive directories.

Once the `archx` script has been run to build thumbnails and indices, use this command:

```
catweb indices/*
```

The wildcard filename "`indices/*`" will expand to a list of all the index files built by `archx`, and these indices, plus the image catalog file `birdimages.xml`, will drive the building of the web pages.

If all goes well, the script will create these files:

1. `index.html` is the start page. This page will contain links to each of the *form pages* described below. These links will be imbedded within an indented, phylogenetic listing, showing all the biological taxa into which the kinds of birds in the catalog fit.
2. In subdirectory "`form/`", one form page will be generated for each biological taxon of which the catalog has at least one image.

3.1. The index page

General ornithological practice is to organize bird records according to taxonomy. Accordingly, the index page will group photos under the various taxa—orders, families, genera, and so on. Fortunately, this project can use the taxonomic infrastructure described in *A system for representing bird taxonomy*⁶. Hence, the general organization of the index page will be an outline-style display of bird taxonomy, with Class Aves the root taxon, and each lower-level taxonomic rank indented one level further.

However, there is no sense in displaying all possible bird taxa. It saves space to display only those taxa for which there is at least one photo.

For each taxon that contains at least one photo, there may be more than one English name that applies. For example, older photos of what is now called Long-tailed Duck may be under the older name, Oldsquaw. The author's preference is to place each unique name under its own form page. In this example, under the taxon (species *Clangula hyemalis*), there would be links to two form pages, one for photos under Long-tailed Duck, and one for Oldsquaw photos. In a future revision, it might be good to add a feature that gives equivalences for older names, so that Oldsquaw photos are automatically moved to Long-tailed Duck.

A first approximation of index page design would be to show a separate heading line for each taxon. Then, under each form name in that taxon that has at least one photo, there would be a link to the form page for that form name. Here is a mockup of the start of the index page, through the geese and swans. The symbol "`→f`" means that the preceding text is a link to the form page for that name.

```
Class Aves: Birds
  Order Anseriformes: Screamers, Swans, Geese and Ducks
    Family Anatidae: Swans, Geese and Ducks
      Subfamily Anserinae: Geese and Swans
        Genus Anser
```

⁶ <http://www.nmt.edu/~shipman/xnomo/>

Species *Anser indicus*: Bar-headed Goose
 Bar-headed Goose →f
 Genus *Chen*
 Species *Chen canagica*: Canada Goose
 Canada Goose →f
 Species *Chen caerulescens*: Snow Goose
 Snow Goose →f
 Form *Chen caerulescens 1*: Blue Goose
 Blue Goose →f

Because the index page may be quite long (many hundreds of lines), there are some obvious steps to compress this report into fewer lines.

- Because species lines always include the genus, there is no point in displaying a separate line for a genus—unless there are photos referred to that genus and no lower (see Section 2, “Definitions” (p. 3) for the meaning of “referred to”).
- When a species has only one form, the species heading could act as the link to the form page, obviating the need for a separate line.

However, when there are multiple form names within a given taxon, the links will appear, one line per form name, under a heading for that taxon.

- Because genus and species names are italicized, there is no point in displaying the titles “Genus” and “Species”.

So, here is our mockup from above, with these improvements.

Class Aves: Birds
 Order Anseriformes: Screamers, Swans, Geese and Ducks
 Family Anatidae: Swans, Geese and Ducks
 Subfamily Anserinae: Geese and Swans
Anser indicus: Bar-headed Goose →f
Chen canagica: Canada Goose →f
Chen caerulescens: Snow Goose →f
Chen caerulescens 1: Blue Goose →f

There is one additional consideration: what if a taxon has only one name referred to it, but that name is not the standard name? For example, if there is only one Long-tailed Duck picture, and it is indexed under Oldsquaw, which of these examples should it look like? In this case, the space-saving choice is to display the standard name, and the name used, on the same line, and make that line a link to the form page. For example:

Clangula hyemalis: Long-tailed Duck (as Oldsquaw) →f

In such a case, the reference name is given first, and the variant as “as ...”.

3.2. The form page

Each form page displays all the cataloged and archived photos that have the same English name. The page's title will be “Shipman's photos of *bird name here*”.

The body of the page is a table showing thumbnails and other data for each image that contains at least one individual of the page's selected form.

The ordering of the images in each section is by descending order of image size. A 35mm frame is roughly 24×36mm, an area of 864 mm². Image areas are expressed as a percentage of the total possible area. For example, an image 12mm×9mm would have an area of 108mm², or 12.5% (108/864 = 0.125).

Columns of the table, left to right:

1. A thumbnail image.
2. Size information. The first line will show the percentage of the image area. The second line will show the width in pixels, and the third line the height in pixels.
3. The catalog number, state and locality, and all additional information from the elements comprising the `info` pattern in the catalog schema⁷: lighting, film, etc.

4. XHTML output

This section describes the XHTML to be produced for the index and form pages.

4.1. XHTML design considerations

It is all very well and good to say that CSS will handle the formatting of the taxonomic headings on the index page, but there is a subtle design problem that must be addressed.

In practice, all my work with bird classification has used the same set of taxonomic ranks: order, family, subfamily, genus, species, and subspecific form. To display taxa as an outline, we could just leave the order lines unindented; indent family lines one increment; indent subfamily lines two increments; and so on.

However, technically, in the underlying *system for representing bird taxonomy*⁸, one can select a different set of ranks.

So the question is how much flexibility to build into the Web design to accommodate different sets of ranks. Also, where should this flexibility reside? To put the latter question differently, what software must change when the set taxonomic ranks changes?

The author's preference is to place the dependency in the CSS stylesheet. The XHTML generation step can tag the taxonomic heading lines with the rank code (e.g., “-f” for subfamily), so the logic that generates the XHTML does not have to be changed when the rank set changes. If the CSS gets out of synch with the rank set, the worst that can happen is the heading won't be formatted correctly; it will, however, still appear. This decision, in the author's opinion, minimizes the impact of using a different rank set.

4.2. XHTML for the index page

Here is the top-level XHTML for the index page. Details of the content of the `body` element are discussed below.

```
<html xmlns='http://www.w3.org/1999/xhtml'>
  <head>
    <title>Shipman's bird photo index</title>
    <link rel='stylesheet' href='birdindex.css' type='text/css' />
  </head>
```

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

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

```

<body>
  <h1>Shipman's bird photo index</h1>
  ...body here...
</body>
</html>

```

The `link` element points at the stylesheet; see Section 5, “Styling with CSS” (p. 9).

The body of the page will consist of a sequence of `div` elements, one for each taxonomic heading, and one for each form name.

The `div` elements for taxonomic headings will be tagged with an attribute “`class='head-X'`”, where *X* is the taxonomic rank code (e.g., ‘g’ for genus or ‘-f’ for subfamily).

Here's an example of a taxonomic heading for Subfamily Anserinae:

```

<div class='head--f'>
Subfamily Anserinae: Geese and Swans
</div>

```

Also, taxonomic names at or below genus rank should be italicized according to the standard biological conventions. The parts to be italicized will be wrapped in a `span` element tagged with `class='genus'`. Here is an example:

```

<div class='head-g'>
<span class='genus'>Lanius</span>
</div>

```

When a higher taxon has at least one photo referred to it, but those photos all use the same name, the heading will also serve as a link to the form page. For example, here is a heading for genus *Empidonax*. Note that form pages live in subdirectory “form/” under the index page, since there may be quite a number of them. For the file naming conventions of form pages, see Section 4.3, “XHTML for the form page” (p. 7).

```

<div class='head-g'>
  <a href='form/empido.html'>
    <span class='genus'>Empidonax</span>
  </a>
</div>

```

Finally, when multiple English names are referred to the same taxon, there will be a line for each form, containing a link to the form page for that name. Such lines are packaged into a `div` element of class `name-line`. Here is an example of a link for a species pair, Downy or Hairy Woodpecker:

```

<div class='name-line'>
  <a href='form/dowwoo-o-haiwoo.html'>
    >Woodpecker, Downy/Woodpecker, Hairy</a>
</div>

```

4.3. XHTML for the form page

Each form page displays all the catalog entries for archived photos that use the same English name.

Because there may be hundreds of form pages, they reside in subdirectory “form/” under the directory where the index page lives.

The name of each form file will be derived from the “bird ID” string corresponding to that name. In most cases, this will be just the six-letter bird code. For example, the page for American Robin will live at “form/amerob.html”.

There are two complications, however. Each possible compound (hybrid or species-pair) form will have its own page. The name of this page will be based on the composite bird code (e.g., “mallar^amewig?”). However, as this example shows, the special characters used in composite bird codes (^|?) have special meanings in Unix shells. So these three characters are translated according to this table:

^	-x-
	-o-
?	-q

Here are some examples:

English name	Code	File name
Ou	ou	form/ou.html
Wigeon, American×Wigeon, European	amewig^eurwig	form/amewig-x-eurwig.html
Flycatcher, Dusky/Flycatcher, Hammond's	dusfly hamfly	form/dusfly-o-hamfly.html
Warbler, Orange-crowned?	orcwar?	form/orcwar-q.html

The bulk of the form page is one large table with three columns:

1. The thumbnail image.
2. The image size, as a percentage of a 24mm×36mm frame.
3. The rest of the width of the table consists of a vertical stack of boxes. The first box contains the catalog number (whose first 10 digits are also the date), the state, and the locality. At this writing, only the **note** element will be shown in the next box. Eventually the other elements such as **desc** and **film** will get their own boxes.

Here is a mockup showing the table and one entry.

```
<table border='4' cellspacing='4'>
  <colgroup>
    <col align='center' />
    <col align='right' valign='top' />
    <col align='left' valign='top' />
  </colgroup>
  <thead>
    <tr>
      <th align='center'>Thumbnail</th>
      <th align='center'>Size</th>
      <th align='center'>Data</th>
    </tr>
  </thead>
  <tbody>
```

Here is the first column of a sample table row. Note the redundant **align** and **valign** attributes. Even though they are the same values from the **colgroup** header, they are repeated here because some contemporary browsers do not properly handle **colgroup**.

```

<tr>
  <td valign='top' align='center'>
    <img src='../thumb/2005-02-18a0005.jpg' alt='thumbnail' />
  </td>

```

The second column displays the image size in two ways: as a percentage of full frame, and using its pixel dimensions. Because the “Size” column has very little information in it, we want to keep that column narrow by stacking these three numbers vertically, using plain `div` elements. The “`×`” character is the multiplication symbol “x”.

```

<td valign='top' align='right'>
  <div>26.37%</div>
  <div>2018</div>
  <div>&#x00d7;2796</div>
</td>

```

The third column contains all the other data. Paragraphs are vertically stacked using `div` elements.

```

<td valign='top' class='cat-info'>
  <div class='ident'>
    <span class='cat-no'>2005-02-18a0005</span>
    NM: Bosque del Apache NWR
  </div>
  <div> <!--Minor data-->
    taking off, wings up, !!
  </div>
</td>
</tr>
</tbody>
</table>

```

Note the links to CSS rules:

div.ident

The box containing the catalog number and location.

span.cat-no

Highlights the catalog number.

td.cat-info

Formats the information block.

See Section 5, “Styling with CSS” (p. 9) for the corresponding CSS rules.

5. Styling with CSS

Here is the style sheet “`birdindex.css`” for the generated pages, as CSS (Cascading Style Sheets) level 2. See the author's reference document, *Styling Web pages with CSS-2*⁹.

First we'll need a basic page style. The author's preference is for insipid earth tones.

`birdindex.css`

```

/* Special stylesheet for the bird index pages.
 * For documentation, see:

```

⁹ <http://www.nmt.edu/tcc/help/pubs/css/>

```

* http://www.nmt.edu/~john/scans/slides/catweb/
*/
body
{ background-color: #ffffdd; /* Pale golden yellow */
  color: #663300; /* Dark brown */
}

```

As discussed in Section 4.1, “XHTML design considerations” (p. 6), we’ll need rules for the `div` elements used to represent the headings for each taxonomic rank. This version of the stylesheet assumes that the rank set in use has these levels:

c	Class
o	Order
f	Family
- f	Subfamily
g	Genus
s	Species
x	Subspecific form

So we’ll need a rule for a `div` elements for each of these classes, where the class name is `head-X`, and `X` is the rank code.

To get proper indentation, we’ll set the `margin-left` property to a larger value for each lower rank. Major ranks will also use a larger text size.

birdindex.css

```

div.head-c
{ font-size: 200%;
}

div.head-o
{ font-size: 160%;
  margin-left: 10px;
  margin-top: 12px;
  border-top: 6px solid;
}

div.head-f
{ font-size: 140%;
  margin-left: 20px;
  margin-top: 8px;
  border-top: 4px solid;
}

div.head--f
{ font-size: 120%;
  margin-left: 30px;
  margin-top: 4px;
  border-top: 2px solid;
}

div.head-g

```

```

{ margin-left: 40px;
}

div.head-s
{ margin-left: 50px;
}

div.head-x
{ margin-left: 60px;
}

```

That takes care of all the possible taxonomic levels. We'll need another rule for rendering the lines for English names. This line must be indented further than all the taxonomic headings.

birdindex.css

```

div.name-line
{ margin-left: 70px;
}

```

The `span.genus` rule is used for italicizing genus and species names.

birdindex.css

```

span.genus
{ font-style: italic;
}

```

The rest of these rules apply to form pages. The `div.ident` rule sets off the box containing the catalog number and location. This rule gives the box a light gray background.

birdindex.css

```

div.ident
{ background-color: #eeeeee;
}

```

The `span.cat-no` rule sets off the catalog number in bold, slightly larger, maroon type.

birdindex.css

```

span.cat-no
{ font-weight: bold;
  font-size: 120%;
  color: maroon;
}

```

The `td.cat-info` rule puts a small amount of padding (three pixels) around the entire table cell in the data column. I could have done this with a `cellpadding` attribute for the whole table, but I wanted the thumbnail to exactly fill its cell.

birdindex.css

```

td.cat-info
{ padding: 3px;
}

```

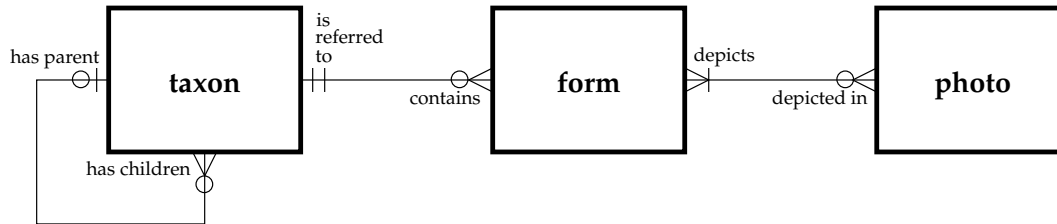
6. Analysis notes

The overall structure of this catalog relates three entities:

1. Each *photo* may depict one or more forms of bird.

2. A *form* refers to all the photos that use a specific English name.
3. The external taxonomy system places each English name into a specific biological *taxon*.

Here is an entity-relationship diagram (ERD) for these three entities. If you are not familiar with ERD notation, see *How to read ERDs*¹⁰.



The **taxon** object has a relationship to itself: a taxon may or may not have a parent, and it may have zero or more children. In short, **taxon** objects are structured as a tree.

Taxa have a one-to-many relationship to forms: a taxon may contain photos for only one form, or for more than one.

Photos have a many-to-many relationship with forms. A single photo may depict more than one kind of bird; and the catalog may contain zero or more photos of a kind of bird.

7. Design notes

The data structure design for this application proceeds directly from the ERD discussed in the previous section. The generated Web pages are basically a taxonomic tree presented in outline form, with links to pages bearing all the photos of a single form.

Hence, the program will build a tree in which each node relates to a taxon, each taxon contains one or more forms, and each form contains one or more photos.

- The tree is made of instances of a class named **TaxonPhotoSet**. Each instance represents all the photos contained in a specific taxon. The root of the tree is the **TaxonPhotoSet** instance for Class Aves. Each instance may have one or more **TaxonPhotoSet** children representing child taxa for which there are photos in the catalog.
- An instance of class **FormPhotoSet** represents all the photos cataloged under a specific form name. A **TaxonPhotoSet** instance is a container for one or more **FormPhotoSet** instances. For example, if the taxon *Clangula hyemalis* contains photos under both the current name (Long-tailed Duck) and the older name (Oldsquaw), the **TaxonPhotoSet** for taxon *Clangula hyemalis* will contain two **FormPhotoSet** objects, one under each name.
- Each photo is represented by an **ArchImage** instance obtained from the image archiving system¹¹. Each **ArchImage** instance describes the size of the image, and contains cataloging information from the image cataloging system¹².

Here is the general flow of the program.

1. Create an empty **TaxonPhotoSet** instance for the root taxon, Class Aves.

This instance will form the root of a taxonomic tree of **TaxonPhotoSet** objects. This tree will contain only nodes for taxa for which there are photos.

¹⁰ http://infohost.nmt.edu/tcc/sa/erd_how.html

¹¹ <http://www.nmt.edu/~john/scans/slides/archx/>

¹² <http://www.nmt.edu/~john/scan/slides/ims/>

- Using the index files named on the command line, work through all the archived, catalogued photos. Each photo may depict multiple forms: each form in the photo is added to the tree separately, once for each form name in the catalog entry. The photo's cataloging information is represented by an `ArchImage` object.

When a photo is added under a given form name, we first find the taxon to which the form is referred. Then add `TaxonPhotoSet` nodes to the tree as necessary to form a path from the root down to the referred taxon—let's call this the *leaf*.

Next we make sure that the leaf `TaxonPhotoSet` contains a `FormPhotoSet` object for the name we're adding. Then we add the photo to that `FormPhotoSet`.

- To generate the Web pages, we traverse the tree of `TaxonPhotoSet` objects in preorder, that is, starting with the root, and recursively visiting the children.

Each `TaxonPhotoSet` node visited is rendered as a `div` element decorated with a class name appropriate to its taxonomic rank.

If the node has photos for only one form, the taxonomic heading will itself be a link to the form page for that form. For taxa with multiple forms, however, each name appears on a separate line, each line a link to the form page for that form.

In either case, the program builds the form page when it builds the link to that page. Each `FormPhotoSet` object corresponds to one form page. The bulk of the content of each form page is the table that displays the thumbnails and other photo data.

8. Code prologue

This section begins the exposition of the actual code, in lightweight literate programming style¹³.

```
#!/usr/local/bin/python
#=====
# catweb: Build index to bird photos. For documentation see:
#   http://www.nmt.edu/~john/scans/slides/catweb/
#-----
```

catweb

8.1. Module imports

Because this application uses generators, the first `import` must be the one that enables generators in older versions of Python.

```
#=====
# Imports
#-----

from __future__ import generators
```

catweb

The `sys` module is used for retrieving command line arguments. We also need the `os` module for manipulating path names.

```
import sys, os
```

catweb

¹³ <http://www.nmt.edu/~shipman/soft/litprog/>

The `txny` module is from *A system for representing bird taxonomy*¹⁴.

catweb

```
import txny as txnyModule
```

We'll also need the `abbr.py` module, which contains the `BirdId` class.

catweb

```
import abbr as abbrModule
```

The `birdimages` module is described in *An XML-based bird cataloging system*¹⁵. It represents cataloged images.

catweb

```
import birdimages
```

The `archindex` module is described in *archx: A program to index a photo archive*¹⁶.

catweb

```
from archindex import *
```

The `xml4create` module is documented in *Python and the XML Document Object Model (DOM) with 4Suite*¹⁷.

catweb

```
import xml4create as xc
```

9. Manifest constants

Here are some constants used in the script, placed at the front for convenience in updating them.

9.1. CSS_STYLESHEET: Stylesheet name

The path to the stylesheet for these pages.

catweb

```
CSS_STYLESHEET = "http://www.nmt.edu/~john/scans/bird/birdindex.css"
```

9.2. FORM_SUBDIR: Form subdirectory name

The name of the subdirectory containing the form pages.

catweb

```
FORM_SUBDIR = "form/"
```

9.3. FULL_FRAME_MM2: Full frame size for 35mm

This constant expresses the area of a full 35mm film frame (nominally 24mm×36mm) in square millimeters.

catweb

```
FULL_FRAME_MM2 = 24 * 36
```

¹⁴ <http://www.nmt.edu/~shipman/xnomo/>

¹⁵ <http://www.nmt.edu/~john/scans/slides/ims/>

¹⁶ <http://www.nmt.edu/~john/scans/slides/archx/>

¹⁷ <http://www.nmt.edu/tcc/help/pubs/pyxml4/>

9.4. INDEX_PAGE: Name of the index page

This is the name of the file where the index page will be written.

catweb

```
INDEX_PAGE = "index.html"
```

9.5. MM_PER_IN: Conversion factor

Number of millimeters per inch.

catweb

```
MM_PER_IN = 25.4
```

10. The main program

The code starts with by setting up the Txny object containing the reference taxonomy, and the root ImageCatalog object containing the descriptions of cataloged images.

catweb

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

def main():
    '''Main program.

       [ index page := taxonomy with links to form pages from
         photo archive indices named on the command line
         those form pages := their content from those indices ]
       ...
    #-- 1 --
    # [ the current directory has a readable, valid aou.xml file ->
    #   txny := a Txny object representing that file ]
    txny = txnyModule.Txny()

    #-- 2 --
    # [ the current directory has a readable, valid
    #   birdimages.xml file ->
    #   catalog := a birdimages.ImageCatalog object
    #           representing that file ]
    catalog = birdimages.ImageCatalog.readFile ( 'birdimages.xml' )
```

We start building the tree of taxa by instantiating a TaxonPhotoSet object linked to the root taxon from txny.

catweb

```
#-- 3 --
# [ rootTaxon := a TaxonPhotoSet for txny.root ]
rootTaxon = TaxonPhotoSet ( txny.root )
```

Now we are ready to process the names of the index files from the command line. Each file describes the contents of one archive; the entries for each file are added to rootTaxon.

catweb

```
#-- 4 --
# [ rootTaxon := rootTaxon with photos from archives named
```

```

#     on the command line, placed into taxa from txny, using
#     forms from catalog ]
for archFileName in sys.argv[1:]:
    addArchive ( txny, catalog, rootTaxon, archFileName )

```

Everything we need to build the web pages comes from > the rootTaxon object.

catweb

```

#-- 5 --
# [ index page := taxonomy with links to form pages from
#     rootTaxon
#   form pages from rootTaxon := content from rootTaxon ]
buildWeb ( rootTaxon )

```

10.1. addArchive(): Process one archive's worth of catalog entries

This function reads the contents of one archive index file, and adds all its catalog information to the rootTaxon object.

catweb

```

# - - -   a d d A r c h i v e   - - -
def addArchive ( txny, catalog, rootTaxon, archFileName ):
    """Add one archive's worth of catalog entries.

    [ (txny is a taxonomy as a txnyModule.Txny) and
      (catalog is a birdimages.ImageCatalog) and
      (rootTaxon is a TaxonPhotoset) and
      (archFileName is a string) ->
        if archFileName names a readable file valid against
        archx.rnc ->
          rootTaxon := rootTaxon with photos from that
                      archive, placed into taxa from txny, using
                      forms from catalog ]

    """

```

The ArchiveIndex.readFile() static method takes care of reading the file and building an ArchiveIndex object to represent it.

catweb

```

#-- 1 --
# [ if archFileName names a file that is readable and valid
#   against archx.rnc ->
#     archIndex := an ArchiveIndex instance representing
#                 that file, with cataloging from catalog
#   else -> raise IOError ]
archIndex = ArchiveIndex.readFile ( catalog, archFileName )

```

We extract each catalog entry in turn and add it to rootTaxon.

catweb

```

#-- 2 --
# [ rootTaxon := rootTaxon with photos added from
#   archIndex, placed into taxa from txny ]
for archImage in archIndex.genArchImages():
    addArchImage ( txny, archImage, rootTaxon )

```

10.2. addArchImage (): Add all forms for one image

This function examines the catalog information in one ArchImage instance, and for each form name cataloged, adds that ArchImage to the tree rooted in rootTaxon.

catweb

```
# - - -   a d d A r c h I m a g e   - - -  
  
def addArchImage ( txny, archImage, rootTaxon ) :  
    """Catalog all forms in one image.  
  
    [ (txny is a txnyModule.Txny) and  
      (archImage is an archindex.ArchImage) and  
      (rootTaxon is a TaxonPhotoSet) ->  
        rootTaxon := rootTaxon with archImage added under  
                    each form cataloged in archImage ]  
  
    """
```

Inside the archImage instance, the .original attribute is a birdimages.Original catalog object. The Original.ab6 attribute is a space-separated list of "bird ID" strings, generally a six-letter code, but possibly a longer code giving information about hybrids, species pairs, and whether the ID was questionable.

We take this string and break it on spaces, converting each one to a txnyModule.BirdId instance that links that form to a specific taxon. Then, for each form, we use the TaxonPhotoSet.addArchImage() method to catalog that image under that taxon and form name.

catweb

```
#-- 1 --  
# [ birdIdList := list of bird codes from  
#     archImage.original.ab6, split at spaces ]  
birdIdList = archImage.original.ab6.split()  
  
#-- 2 --  
# [ rootTaxon := rootTaxon with archImage added under  
#     each form code from birdIdList ]  
for rawBirdId in birdIdList:  
    #-- 2 body --  
    # [ if rawBirdId is defined in txny ->  
    #     rootTaxon := rootTaxon with archImage added  
    #                 under rawBirdId ]  
  
    #-- 2.1 --  
    # [ if rawBirdId is defined in txny ->  
    #     birdId := an abbrModule.BirdId instance representing  
    #               rawBirdId  
    #     else -> raise Exception ]  
    try:  
        birdId = abbrModule.BirdId.parse ( txny, rawBirdId )  
    except Exception, detail:  
        print >>sys.stderr, ( "*** Invalid bird ID '%s': %s" %  
                               (rawBirdId, detail) )  
  
        continue  
  
#-- 2.2 --
```

```

# [ rootTaxon := rootTaxon with archImage added under birdID ]
try:
    rootTaxon.addArchImage ( birdId, archImage )
except KeyError, detail:
    print >>sys.stderr, "****", detail

```

10.3. buildWeb () : Build all Web pages

This method generates the index page and all form pages.

catweb

```

# - - -   b u i l d W e b   - - -

def buildWeb ( rootTaxon ):
    """Build all web pages.

    [ rootTaxon is a TaxonPhotoSet ->
      index page := taxonomy with links to form pages from
      rootTaxon
      form pages from rootTaxon := content from rootTaxon ]
    """

```

First we use the `webPage ()` function to set up what will become the index page.

catweb

```

#-- 1 --
# [ indexPage := a new XHTML page as an xc.Document ]
#   body := the body element of that page ]
indexPage, body = webPage ( "Shipman's bird photo index",
                           CSS_STYLESHEET )

```

The content of the page is added by visiting each node in the taxonomic tree made up of `TaxonPhotoSet` instances. The `.walk ()` method on the root `TaxonPhotoSet` instance takes care of this visitation process. It yields each node of the tree in sequence, using pre-order traversal (that is, the root node comes first, followed recursively by its child subtrees).

At each node, we call the node's `.buildPage ()` method to add the node's content to the index page, including generation of all the form pages for forms referred to that node's taxon.

catweb

```

#-- 2 --
# [ body := body with content added for all nodes in the
#   tree of TaxonPhotoSet instances rooted at rootTaxon ]
for taxonSet in rootTaxon.walk():
    #-- 2 body --
    # [ taxonSet is a TaxonPhotoSet instance ->
    #   body := body with content added from node
    #   form pages for forms in node := content from
    #   forms in node ]
    taxonSet.buildPage ( body )

```

Now that the index page has been built, write it out.

catweb

```

#-- 3 --
# [ if index page can be created anew ->

```

```
#    index page := indexPage, serialized ]
indexFile = open ( INDEX_PAGE, "w" )
indexPage.write ( indexFile )
indexFile.close()
```

10.4. webPage () : Create an empty XHTML page

This function builds a generic page of XHTML, to which other functions can add content later. It returns a 2-tuple (*doc*, *body*): *doc* is an `xc.Document` instance containing the whole page, and *body* is the body element of that page as an `xc.Element` instance.

catweb

```
# - - -   w e b P a g e   - - -

def webPage ( titleText, stylesheet=None ) :
    """Builds the skeleton of an XHTML page.

    [ (titleText is the text for the title and h1 elements) and
      (stylesheet is the href of a stylesheet, or None) ->
        return (a new XHTML page with that title and stylesheet
                as an xc.Document, the body element of that page as an
                xc.Element) ]
    """
```

Document creation is described in the documentation for the `xml4create` module¹⁸. We'll even put in a DOCTYPE so the W3C validator¹⁹ will approve.

catweb

```
#-- 1 --
# [ doc := a new XHTML 1.0 Strict page as an xc.Document ]
doctype = xc.DocumentType ( "html",
                            "-//W3C//DTD XHTML 1.0 Strict//EN",
                            "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" )
doc = xc.Document ( "html", doctype )
```

The rest is very similar to the example given in the `xml4create` documentation.

catweb

```
#-- 2 --
# [ doc := doc with a new 'head' child containing
#       titleText as its text, and a 'body' child
#       containing titleText as its heading
#   head := that 'head' child
#   body := that 'body' child ]
head = xc.Element ( doc.root, "head" )
title = xc.Element ( head, "title" )
xc.Text ( title, titleText )
body = xc.Element ( doc.root, "body" )
h1 = xc.Element ( body, "h1" )
xc.Text ( h1, titleText )

#-- 3 --
```

¹⁸ <http://infohost.nmt.edu/tcc/help/pubs/pyxml4/xml4create.html>

¹⁹ <http://validator.w3.org>

```

# [ if stylesheet is None -> I
#   else ->
#     head := head with a new 'link' child linking to
#           the stylesheet named (stylesheet) ]
if stylesheet is not None:
    link = xc.Element ( head, 'link', href=stylesheet,
                        rel='stylesheet', type='text/css' )

#-- 4 --
return (doc, body)

```

11. class TaxonPhotoSet: The taxonomic tree node

Each instance of this class is a container for all the photos that depict members of a specific taxon. Instances are arranged in the form of a tree. The root node is a container for the entire catalog, and represents the root taxon of birds, Class Aves.

Here is the interface. The constructor takes one argument: the taxon related to this node.

catweb

```

# - - - - - c l a s s   T a x o n P h o t o S e t   - - - - -

class TaxonPhotoSet:
    """Represents all photos contained in a specific taxon.

    Exports:
    TaxonPhotoSet ( taxon ):
        [ taxon is a txny.Taxon instance ->
          return a new, empty TaxonPhotoSet for that taxon ]
    .taxon:      [ as passed to constructor; read-only ]
    .nContained: [ number of photos in self's subtree ]
    .nReferred:  [ number of photos referred to self's taxon ]

```

The `.addArchImage()` method is used to load catalog entries into the tree. For each photo, this method is called once for each kind of bird in the photo.

Note the precondition that the specified `birdId` must be referred to a taxon that is either self or a descendant of self. It simplifies the logic of adding a new photo to know that the photo goes somewhere in the correct subtree. It is no burden on the caller who adds the photo to the root of the tree, since all birds are in class Aves. See Section 11.2, “TaxonPhotoSet.addArchImage(): Add one photo” (p. 22).

catweb

```

.addArchImage ( birdId, archImage ):
    [ birdId is referred to a taxon contained in self.taxon ->
      (birdId is a composite bird form code as an
       abbr.BirdId object) and
      (archImage is an image catalog entry as a
       archindex.ArchImage instance) ->
      self := self with archImage added under birdId ]

```

The `.walk()` method walks the tree, starting at the given node, and then recursively visiting its descendants. This method is called to generate the various taxonomic headings on the index page. See Section 11.5, “TaxonPhotoSet.walk(): Walk the tree” (p. 25).

```
.walk():
    [ generate self, followed by the descendants of self,
      in preorder traversal ]
```

Because the presentation of a taxon with only one form is different than the presentation of a multi-form taxon, the `.nForms()` method allows the caller to determine the number of forms. See Section 11.6, “`TaxonPhotoSet.nForms()`: Number of contained forms” (p. 26).

```
.nForms():    [ returns the number of forms in self ]
```

The `.genForms()` method generates the contained `FormPhotoSet` objects in order. See Section 11.7, “`TaxonPhotoSet.genForms()`: Generate referred `FormPhotoSet` objects” (p. 26).

```
.genForms():
    [ generate the FormPhotoSet objects in self, in
      ascending order by inverted English name ]
```

Here are the internals of the instance. Each node maintains two counters. The `.nContained` attribute counts all the photos ever added to this node's subtree. The `.nReferred` attribute is a count of only those photos referred to this node's taxon.

Each `TaxonPhotoSet` instance is a container for two kinds of objects: child taxa, and `FormPhotoSet` instances for any photos that may be referred to this taxon.

One of the author's favorite standard Python tricks serves nicely to structure both these container relationships: we keep the contained values in a dictionary, and access them using a key that happens to sort in the desired access order.

The standard Python trick is: use the dictionary's `.keys()` method to extract a list of keys; sort that list; and then go through the dictionary in order using that list.

Dictionary `.childMap` contains the child `TaxonPhotoSet` elements as values. In order to access the children in taxonomic order, we use the `txKey` attribute of the taxon as the key, which is a string of digits that orders taxa in phylogenetic order.

```
State/Internals:
    __childMap:
        [ a dictionary whose values are the child TaxonPhotoSet
          nodes of self, and each corresponding key is the
          .txKey attribute of self's taxon ]
```

The desired order of form names within a taxon is alphabetical order by the inverted English name.

```
    __formMap:
        [ a dictionary whose values are the FormPhotoSet
          objects holding photos referred to self.taxon,
          and each corresponding key is the inverted
          English name for that form ]
    """
```

11.1. TaxonPhotoSet.__init__(): Constructor

The constructor does not do much. It stores the specified `taxon`, zeroes the counts of contained photos, and sets up the internal data structures.

catweb

```
# - - -   T a x o n P h o t o S e t . _ _ i n i t _ _   - - -  
  
def __init__ ( self, taxon ):  
    """Constructor for TaxonPhotoSet"""  
    self.taxon = taxon  
    self.nContained = 0  
    self.nReferred = 0  
    self.__childMap = {}  
    self.__formMap = {}
```

11.2. TaxonPhotoSet.addArchImage(): Add one photo

Because multiple forms of birds may appear in one photo, each photo may be added under multiple names. In that case, the `.addArchImage()` method is called once for each name.

Hence, this method operates on not just a catalog entry (as an `ArchImage` object from the `archindex.py` module), but also on a `BirdId` object (from the `abbr.py` module) that represents the kind of bird.

catweb

```
# - - -   T a x o n P h o t o S e t . a d d A r c h I m a g e   - - -  
  
def addArchImage ( self, birdId, archImage ):  
    """Add one photo to the report under a given birdId  
    """
```

Adding a new photo to the structure always starts at the root node, by calling its `.addArchImage()` method. In general, this operation is recursive. It may involve the addition of multiple new taxa to the tree, starting at the top and terminating at the taxon to which the photo is referred.

Depending on whether the given `birdId` is referred to `self`'s taxon or not, there are two cases.

- Basis case: if the `birdId` is referred to `self`'s taxon, no new child nodes need to be added under this one. First, make sure that `self` has a `FormPhotoSet` for this bird name, then add the photo to that `FormPhotoSet`.
- Recursive case: if the `birdId` is referred to a descendant of `self`, first insure that nodes are added to `self`'s subtree down to the referred taxon's node, then add the photo to that node.

The first order of business is to separate the two cases. The `taxon` attribute of a `BirdId` object is the taxon to which that form is referred. The comparison function (`.__cmp__()`) in the `Taxon` class compares instances according to their `txKey` attribute, so we can use the ordinary Python “==” comparison to see if they are the same taxon.

For the internal methods that handle the two cases, see Section 11.3, “`TaxonPhotoSet.__addReferred()`: Basis case” (p. 23) and Section 11.4, “`TaxonPhotoSet.__addContained()`: Recursive case” (p. 24).

catweb

```
# - - 1 - -  
# [ if self.taxon is the same as birdId.taxon ->
```

```

#     self.nReferred += 1
#     self.__formMap := self.__formMap with archImage added
#         under the name from birdId
# else ->
#     self.__childMap := self.__childMap with new descendant
#         nodes added as necessary down to birdId.taxon,
#         and archImage added under the name from birdId to
#         the node for birdId.taxon ]
if self.taxon == birdId.taxon:
  #-- 1.1 --
  self.__addReferred ( birdId, archImage )
  self.nReferred += 1
else:
  #-- 1.2 --
  self.__addContained ( birdId, archImage )

```

The final step is bookkeeping: counting the number of photos in this subtree.

catweb

```

#-- 2 --
self.nContained += 1

```

11.3. TaxonPhotoSet.__addReferred(): Basis case

This method adds the new photo to the forms in self.

catweb

```

# - - -   T a x o n P h o t o S e t . _ _ a d d R e f e r r e d   - - -
def __addReferred ( self, birdId, archImage ):
  """Add a catalog entry to one of the FormPhotoSets in self.

  [ (birdId is a BirdId referred to self.taxon) and
    (archImage is a catalog entry as an archindex.ArchImage) ->
    self.__formMap := self.__formMap with archImage added
                    under the name from birdId ]
  """

```

The first step is to make sure there is a FormPhotoSet in self.__formMap under the name used by birdId. See Section 12, “class FormPhotoSet: All photos for one name” (p. 33).

catweb

```

#-- 1 --
# [ formName := inverted English name from birdId ]
formName = birdId.engComma()

#-- 2 --
# [ if self.__formMap has a key formName ->
#   formSet := the corresponding value
# else ->
#   self.__formMap := a new FormPhotoSet instance
#                   for formName
#   formSet := that new FormPhotoSet instance ]
try:

```

```

        formSet = self.__formMap[formName]
    except KeyError:
        formSet = FormPhotoSet ( birdId )
        self.__formMap[formName] = formSet

```

At this point, formSet is the FormPhotoSet instance to which the new photo is to be added. See Section 12.2, "FormPhotoSet.addArchImage(): Add one photo" (p. 34).

catweb

```

#-- 3 --
# [ formSet := formSet with archImage added ]
formSet.addArchImage ( archImage )

```

11.4. TaxonPhotoSet.__addContained(): Recursive case

This method is used when the form name to be added is not referred to self's taxon. It recursively adds new nodes to self's subtree down to the level where the form name is referred, at which point the basis case adds the photo at that level.

catweb

```

# - - - T a x o n P h o t o S e t . _ _ a d d C o n t a i n e d - - -
def __addContained ( self, birdId, archImage ):
    """Recursive case: add a photo to a descendant taxon.

    [ (birdId is a BirdId referred to a descendant of
      self.taxon) and
      (archImage is a catalog entry as an archindex.ArchImage) ->
      self.__childMap := self.__childMap with new descendant
      nodes added as necessary down to birdId.taxon,
      and archImage added under the name from birdId to
      the node for birdId.taxon ]

    """

```

Each recursive call first ensures that the tree has a node for the child of self containing the new form name, and then it adds the new catalog entry to the child.

The first task is to determine which child taxon contains the new form name. It might be an actual child of self, or it might be an ancestor further down the tree.

catweb

```

#-- 1 --
newTaxon = birdId.taxon

#-- 2 --
# [ newTaxon is a descendant of self.taxon ->
#   childTaxon := the child of self.taxon that
#   contains newTaxon ]
childTaxon = self.taxon.childContaining ( newTaxon )

```

Next we make sure that self.childMap has a child node for this child.

catweb

```

#-- 3 --
# [ if self.__childMap has a key for childTaxon.txKey ->

```

```

#   childNode := the corresponding value
#   else ->
#   self.__childMap [ childTaxon.txKey ] := a new
#       TaxonPhotoSet for childTaxon
#   childNode := that same TaxonPhotoSet ]
try:
    childNode = self.__childMap [ childTaxon.txKey ]
except KeyError:
    childNode = TaxonPhotoSet ( childTaxon )
    self.__childMap [ childTaxon.txKey ] = childNode

```

A recursive call to the child node's `.addArchImage()` method will add the new photo to the correct subtree. See Section 11.2, “`TaxonPhotoSet.addArchImage()`: Add one photo” (p. 22).

catweb

```

#-- 4 --
# [ childNode := childNode with archImage added under
#     birdId ]
childNode.addArchImage ( birdId, archImage )

```

11.5. `TaxonPhotoSet.walk()`: Walk the tree

This method recursively walks the tree, yielding the current node, followed by its subtrees.

catweb

```

# - - -   T a x o n P h o t o S e t . w a l k   - - -

def walk ( self ):
    """Recursive tree walker, yielding nodes in pre-order.
    """

```

The current node is generated first.

catweb

```

#-- 1 --
yield self

```

This method specifies that the children are generated in taxonomic order. Since the keys in the `self.__childMap` dictionary are in phylogenetic order, we extract the key set, sort it, and then recursively generate the subtrees in order by that set.

catweb

```

#-- 2 --
# [ childKeys := a list containing the keys of
#     self.__childMap in ascending order ]
childKeys = self.__childMap.keys()
childKeys.sort()

#-- 3 --
# [ childKeys is a list of keys in self.__childMap ->
#     generate the nodes from subtrees rooted in the nodes
#     that are values of self.__childMap, in childKeys order ]
for txKey in childKeys:
    childNode = self.__childMap[txKey]

```

```
    for node in childNode.walk():
        yield node
```

Raising `StopIteration` terminates the generator.

catweb

```
    #-- 4 --
    raise StopIteration
```

11.6. `TaxonPhotoSet.nForms()`: Number of contained forms

This method returns the number of forms referred to `self`. Since the dictionary `self.__formMap` has one entry for each referred form, we need only return the cardinality of that dictionary.

catweb

```
# - - - T a x o n P h o t o S e t . n F o r m s - - -

def nForms ( self ):
    """How many form names are referred to self?
    """
    return len ( self.__formMap )
```

11.7. `TaxonPhotoSet.genForms()`: Generate referred `FormPhotoSet` objects

This method generates the `FormPhotoSet` objects contained in `self`, in ascending order by their inverted English name. The code is straightforward: extract the English names that are the keys in the `.__formMap` dictionary, sort them, and generate the values from the dictionary in that order.

catweb

```
# - - - T a x o n P h o t o S e t . g e n F o r m s - - -

def genForms ( self ):
    """Generate self's referred forms.
    """

    #-- 1 --
    # [ keyList := the keys from self.__formMap, sorted ]
    keyList = self.__formMap.keys()
    keyList.sort()

    #-- 2 --
    # [ keyList is a list of keys in self.__formMap ->
    #   generate the values from self.__formMap in the
    #   order specified by keyList ]
    for key in keyList:
        yield self.__formMap[key]

    #-- 3 --
    raise StopIteration
```

11.8. TaxonPhotoSet.buildPage(): Construct XHTML

This method is called to render one taxon's content on the index page, and also to build all form pages (and links to them) for forms referred to this taxon.

catweb

```
# - - - T a x o n P h o t o S e t . b u i l d P a g e - - -

def buildPage ( self, parent ) :
    """Add one taxon to the index page, and build all its form pages

    [ parent is an xc.Element node ->
      parent := parent with XHTML child nodes added to
                represent self
      form pages for forms referred to self's taxon :=
                content displaying thumbnails and catalog info
                for forms in self ]

    """
```

In one case, this method builds nothing at all: if it is a genus-level taxon that has no forms referred to it. See Section 3.1, "The index page" (p. 4) for remarks on this case.

catweb

```
#-- 1 --
# [ if self is a genus-level taxon with no forms referred
#   to it ->
#     return
#   else -> I ]
if ( ( self.taxon.rank.depth ==
      self.taxon.txny.hier.genusRank().depth ) and
     ( len(self.__formMap) == 0 ) ):
    return
```

All the content for this taxon will be wrapped in a div element of class head-*R*, where *R* is the code for the taxonomic rank of self's taxon.

catweb

```
#-- 2 --
# [ parent := parent with a div added, tagged with
#   the heading class for self's taxonomic rank
#   div := that new div element ]
div = xc.Element ( parent, "div" )
div["class"] = "head-%s" % self.taxon.rank.code
```

Here we divide the logic into three cases.

- If no form is referred to this taxon, the name is added directly to the `div`.
- If there is only one form referred to this taxon, the name is a link to the form page.
- If there are multiple forms referred to this taxon, the taxon's name is not a link, but it is followed by lines linking to the various form pages.

If there is a single form in this taxon, the content is the taxonomic heading as a link to the form page for that form. If there are multiple forms, the content starts with the taxonomic heading, followed by one additional `div` element for each form's name as a link to the corresponding form page.

```

#-- 3 --
# [ if len(self.__formMap) == 0 ->
#   div := div + (self's inverted name)
# else if len(self.__formMap) <= 1 ->
#   div := div + (link to form page containing self's
#                 inverted name, and the form name if
#                 different than self's inverted name
#                 form page for self.__formMap's value := content
#                 displaying thumbnails and catalog info for
#                 self.__formMap's value
# else ->
#   div := div + (self's inverted name) +
#               (links to form pages for forms in self.__formMap)
#   form pages for self.__formMap values := content
#   displaying thumbnails and catalog info for
#   values in self.__formMap ]
if len(self.__formMap) == 0:
    self.__webName ( div )
elif len(self.__formMap) == 1:
    self.__buildSingleForm ( div )
else:
    self.__buildMultiForms ( div )

```

11.9. TaxonPhotoSet.__buildSingleForm(): The taxon has only one form

This method adds the content for one taxon to the index page when there is only one form referred to that taxon. It also builds the form page for that form.

```

# - - - T a x o n P h o t o S e t . _ _ b u i l d S i n g l e F o r m

def __buildSingleForm ( self, div ):
    """Build index and form content for the single-form case.

    [ (self.__formMap has only one entry) and
      (div is an xc.Element) ->
        div := div + (link to form page containing self's
                      inverted name, and the form name if
                      different than self's inverted name
                      form page for self.__formMap's value := content
                      displaying thumbnails and catalog info for
                      self.__formMap's value ]
    """

```

To review the XHTML build in this case, refer to Section 4.2, “XHTML for the index page” (p. 6). The subtree we generate here is rooted in an “a” (hyperlink) element whose href attribute points at the form page. The link text is the scientific name, followed by the inverted English name. If the inverted English name of the sole form differs from the taxon's standard English name, we add the content “(as *otherName*)”.

First, build the link element.

```

#-- 1 --
# [ div := div + (a new "a" element)
#   link := that "a" element ]
link = xc.Element ( div, "a" )

```

Next we extract the sole entry in `self.__formMap`. The `.keys()` method on a dictionary produces a list of keys, a singleton in our case.

```

#-- 2 --
# [ formSet := the sole value in self.__formMap ]
formKey = self.__formMap.keys()[0]
formSet = self.__formMap[formKey]

```

At this point, `formSet` is a `FormPhotoSet` instance, and its `.pathName()` method returns the pathname to its form page (which hasn't been built yet).

```

#-- 3 --
# [ formSet is a FormPhotoSet ->
#   link := link with an href attribute added whose
#         value is the path name to formSet's form page ]
link [ "href" ] = formSet.pathName()

```

Add, as link text, the scientific and English names for `self`'s taxon. Then, if the form's inverted English name is different, add the "(as *otherName*)" content.

```

#-- 4 --
# [ link := link + (text of self's names) ]
self.__webName ( link )

#-- 5 --
# [ if self.taxon's inverted English name differs from
#   formSet's inverted English name ->
#   link := link + "(as " + (formSet's inverted
#                             English name) + ")"
#   else -> I ]
formEng = formSet.birdId.engComma()
if formEng != self.taxon.engComma:
    xc.Text ( link, "(as %s)" % formEng )

```

All that remains is to generate `formSet`'s form page.

```

#-- 6 --
# [ form page for formSet := content displaying
#   thumbnails and catalog data for formSet ]
formSet.buildPage()

```

11.10. `TaxonPhotoSet.__webName()`: Fill in scientific and English names

This method builds the XHTML for the taxon's scientific name and, if defined, its inverted English name. For design notes, see Section 4.2, "XHTML for the index page" (p. 6).

```
# - - - T a x o n P h o t o S e t . _ _ w e b N a m e - - -
def __webName ( self, parent ):
    """Add the scientific and English names to a Web page.

    [ parent is an xc.Element ->
      parent := parent + (self.taxon's scientific name) +
                    (self.taxon's English name, if known) ]
    """
```

The principal complication is that we must italicize genus-level and lower-rank names by wrapping them in a `span class="genus"` element; see Section 5, "Styling with CSS" (p. 9). The `depth` attribute of a taxonomic rank is 0 for class, 1 for order, and so on, so we can test the rank depth of `self.taxon` against the depth of the genus rank to determine whether to italicize.

```
#-- 1 --
# [ selfDepth := taxonomic rank depth of self.taxon
#   genusDepth := taxonomic rank depth of the genus
#     rank of the taxonomy used by self.taxon ]
selfDepth = self.taxon.rank.depth
genusDepth = self.taxon.txny.hier.genusRank().depth
```

If this rank is genus or deeper, we now add the `span` element, otherwise we don't. In either case we set the variable `textParent` to point to the node to which the name text is to be added.

```
#-- 2 --
# [ if selfDepth >= genusDepth ->
#   parent := parent + (a new span element with
#     class="genus")
#   textParent := that new span element
# else ->
#   textParent := parent ]
if selfDepth >= genusDepth:
    textParent = xc.Element ( parent, "span",
                              class_="genus" )
else:
    textParent = parent
```

Next, the scientific name is added to `textParent`. We add the rank name to ranks above genus level (e.g., "Family Fringillidae"), but omit it for genera and deeper because the italicization tells the reader it's a scientific name.

```
#-- 3 --
# [ textParent := textParent + (self.taxon's scientific
#   name) ]
if selfDepth < genusDepth:
    xc.Text ( textParent, "%s " % self.taxon.rank.name )
xc.Text ( textParent, self.taxon.sci )
```

Finally, display the English name if there is one. The `txny.py` module sets the `.eng` attribute to the same as the `.sci` attribute by default, so in that case, don't redundantly display it. This text is added to `parent`, not `textParent`, so that it will be outside the `span` if present.

```

#-- 4 --
# [ if self.taxon.eng != self.taxon.sci ->
#     parent := textParent + ": " + self.taxon.engComma
if self.taxon.eng != self.taxon.sci:
    xc.Text ( parent, ": %s" % self.taxon.engComma )

```

11.11. TaxonPhotoSet.__buildMultiForms(): The taxon has multiple forms

This method generates content for one taxon on the index page for the case where more than one form name is referred to that taxon.

```

# - - -   T a x o n P h o t o S e t . _ _ b u i l d M u l t i F o r m s

def __buildMultiForms ( self, parent ):
    """Build index and form content for the multi-form case.

    [ parent is an xc.Element ->
      parent := parent + (self's inverted name) +
        (links to form pages for forms in self.__formMap)
      form pages for self.__formMap values := content
        displaying thumbnails and catalog info for
        values in self.__formMap ]
    """

```

First we add the form name to the parent div.

```

#-- 1 --
# [ parent := parent with self's name added as text ]
self.__webName ( parent )

```

The forms referred to this taxon will be displayed in ascending order by inverted English name. This is achieved using the standard Python trick of keeping instances as values in a dictionary, and using a key that sorts in the desired order—in this case, the inverted English name.

```

#-- 2 --
# [ nameList := keys of self.__formMap, sorted ]
nameList = self.__formMap.keys()
nameList.sort()

```

For each key in nameList, the corresponding value is a FormPhotoSet instance. The generation of the index page content, and corresponding photo page, for each form is done by Section 11.12, “Taxon-PhotoSet.__buildFormLine(): Build one form link and form page” (p. 32).

```

#-- 3 --
# [ parent := parent with divs added containing the
#     names from self.__formMap in order by nameList,
#     each as a link to the corresponding value from
#     self.__formMap
#     form pages for values in self.__formMap := content

```

```

#     from those values ]
for formName in nameList:
  #-- 3 body --
  # [ let formSet == self.__formMap[formName]
  #   in
  #     parent := parent with a div added containing the
  #             name from formSet and a link to formSet's form page
  #     form page for formSet := content from formSet ]
  self.__buildFormLine ( parent, self.__formMap[formName] )

```

11.12. TaxonPhotoSet.__buildFormLine(): Build one form link and form page

This method builds the div on the index page containing the link to a form page. It also creates the corresponding form page.

catweb

```

# - - - T a x o n P h o t o S e t . _ _ b u i l d F o r m L i n e

def __buildFormLine ( self, parent, formSet ):
    """Build one form page link and the corresponding form page.

    [ (parent is an xc.Element) and
      (formSet is a FormPhotoSet) ->
      parent := parent with a div added containing the
              name from formSet and a link to formSet's form page
      form page for formSet := content from formSet ]
    """

```

The XHTML generated here is described in Section 4.2, "XHTML for the index page" (p. 6). The only child added to parent is the div element, with class="name-line",

catweb

```

#-- 1 --
# [ parent := parent with a new div element added
#   div := that new div element ]
div = xc.Element ( parent, "div", class_="name-line" )

```

The only child of the div is the link ("a") element that points to the form page.

catweb

```

#-- 2 --
# [ div := div with a new 'a' element added whose href
#   attribute is the relative URL of formSet's page
#   link := that 'a' element ]
link = xc.Element ( div, 'a', href=formSet.pathName() )

```

The link text is the form's inverted English name.

catweb

```

#-- 3 --
# [ link := link with formSet's inverted English name
#   added as link text ]
xc.Text ( link, formSet.birdId.engComma() )

```

Having finished generating all the index page content for this form, now generate the actual form page.

```
#-- 4 --
# [ form page for formSet := content from formSet ]
formSet.buildPage()
```

12. class FormPhotoSet: All photos for one name

Each instance of a FormPhotoSet contains all the images that have the same inverted English name.

```
# - - - - - c l a s s   F o r m P h o t o S e t   - - - - -

class FormPhotoSet:
    """Container for photos that have birds with a given English name.

    Exports:
    FormPhotoSet ( birdId ):
        [ birdId is a bird form code as an abbr.BirdID ->
          return a new, empty FormPhotoSet for birdId ]
    .birdId:      [ as passed to constructor; read-only ]
    .addArchImage ( archImage ):
        [ archImage is a photo catalog entry as an
          archindex.ArchImage instance ->
          self := self with archImage added ]
    .genArchImages():
        [ generate the ArchImage instances in self, in
          descending order by image size, with catalog number
          as a secondary key ]
    .pathName():
        [ returns the relative path name to the page that
          displays this form ]
    .buildPage():
        [ form page for self := content displaying
          thumbnails and catalog data for self ]
```

One internal data structure suffices to hold all the ArchImage instances of the set. One of the author's standard Python tricks is to keep a set of values in a dictionary, and then structure the key so that, when sorted, it produces the values in the desired order.

As discussed in Section 3.2, "The form page" (p. 5), the principal sort order of images is in descending order by image area. In case two images happen to have the same area, their catalog number is used as a tie-breaker. Hence, the dictionary key is a tuple of two values:

```
(-area, catNo)
```

where `-area` is the image's area in square millimeters as a float, negated, and `catNo` is the image's catalog number.

```
State/Invariants:
    .__sizeMap:
        [ a dictionary whose values are the ArchImage instances
          contained in self, and each key is a 2-tuple whose
```

```

        first element is the image's area in square mm as a
        float, negated, and the second element is the
        image's catalog number ]
    """

```

12.1. FormPhotoSet.__init__(): Constructor

All the constructor does is to record its arguments and initialize the `.__sizeMap` dictionary.

catweb

```

# - - -   F o r m P h o t o S e t . _ _ i n i t _ _   - - -

def __init__ ( self, birdId ):
    """Constructor."""
    self.birdId = birdId
    self.__sizeMap = {}

```

12.2. FormPhotoSet.addArchImage(): Add one photo

To add a new `ArchImage` object to this `FormPhotoSet` instance, we construct the 2-tuple used as a key to `self.__sizeMap`, then store the `ArchImage` under that key. There shouldn't be any duplications, but let's check anyway.

catweb

```

# - - -   F o r m P h o t o S e t . a d d A r c h I m a g e   - - -

def addArchImage ( self, archImage ):
    """Add a new archived, cataloged image to self.
    """
    #-- 1 --
    # [ area := number of pixels in archImage ]
    area = archImage.wide * archImage.high

    #-- 2 --
    # [ key := ( -area, catalog number from archImage ) ]
    key = ( -area, archImage.original.catNo )

    #-- 3 --
    if self.__sizeMap.has_key ( key ):
        raise KeyError, ( "Duplicate form photo: area %dpx, "
            "catalog number %s" %
            (area, archImage.original.catNo) )
    else:
        self.__sizeMap [ key ] = archImage

```

12.3. FormPhotoSet.genArchImages(): Generate contained photos

This method generates all our contained `ArchImage` instances. It uses the standard Python trick of setting up a dictionary key so that it sorts in the desired order. In our case, the order is descending order by area, with catalog number as a tiebreaker.

```
# - - -   F o r m P h o t o S e t . g e n A r c h I m a g e s   - - -

def genArchImages ( self ):
    """Generate contained images.
    """
    keyList = self.__sizeMap.keys()
    keyList.sort()
    for key in keyList:
        yield self.__sizeMap[key]
    raise StopIteration
```

12.4. FormPhotoSet.pathName(): Form page relative path name

This method returns the path name of this form's form page, relative to the index page. Python's `os.path.join()` function is used to mate the directory and file names. The file name is based on the bird code from `self.birdId`, but we must sanitize it by translating the relationship codes. The class variable `relMap` maps those codes onto strings acceptable for use in file names.

```
# - - -   F o r m P h o t o S e t . p a t h N a m e   - - -

relMap = { abbrModule.REL_HYBRID: "-x-",
            abbrModule.REL_PAIR:  "-o-" }

def pathName ( self ):
    """Returns the relative path name of self's form page.
    """

    #-- 1 --
    # [ if self.birdId.rel is None ->
    #     basename := self.birdId.abbr
    # else ->
    #     basename := self.birdId.abbr +
    #                 (self.birdId.rel, translated using self.relMap) +
    #                 self.birdId.abbr2 ]
    if self.birdId.rel is None:
        basename = self.birdId.abbr.rstrip().lower()
    else:
        cleanRel = self.relMap [ self.birdId.rel ]
        basename = ( "%s%s%s" %
                    (self.birdId.abbr.rstrip().lower(), cleanRel,
                     self.birdId.abbr2.rstrip().lower()) )
```

To distinguish questionable forms, we add "-q" to the end in that case.

```
#-- 2 --
if self.birdId.q: suffix = "-q"
else:            suffix = ""
```

For the name of the forms subdirectory, see Section 9.2, "FORM_SUBDIR: Form subdirectory name" (p. 14).

```
#-- 2 --
# [ return FORM_SUBDIR + basename + ".html" ]
path = os.path.join ( FORM_SUBDIR, basename )
return "%s%s.html" % (path, suffix)
```

12.5. FormPhotoSet.buildPage(): Build the XHTML page for one form

This method is the driver for the construction of the XHTML page representing the forms in this FormPhotoSet.

```
# - - -   F o r m P h o t o S e t . b u i l d P a g e   - - -

def buildPage ( self ):
    """Builds the XHTML page for self.
    """
```

For the URL of the stylesheet, see Section 9.1, “CSS_STYLESHEET: Stylesheet name” (p. 14).

```
#-- 1 --
# [ page := a new, generic XHTML page with self's title
#       body := the body element of that page ]
titleText = ( "Shipman's bird photo index: %s" %
              self.birdId.engComma() )
page, body = webPage ( titleText, CSS_STYLESHEET )
```

The body of the page is mainly a table. See Section 12.6, “formSet.__buildTable(): Build the image table” (p. 37).

```
#-- 2 --
# [ body := body with a table added containing
#       thumbnails and catalog data from self ]
self.__buildTable ( body )
```

The page is now complete. Create the file and serialize the page to it.

```
#-- 3 --
# [ if self's page can be created new ->
#   pagePath := path to self's file
#   pageFile := a new, empty file ]
pagePath = self.pathName()
pageFile = open ( pagePath, "w" )

#-- 4 --
# [ pageFile += page, serialized ]
page.write ( pageFile )
pageFile.close()
```

12.6. formSet.__buildTable(): Build the image table

This method builds the table that holds the thumbnails and image data. For the XHTML, see Section 4.3, "XHTML for the form page" (p. 7).

catweb

```
# - - -   F o r m P h o t o S e t . _ _ b u i l d T a b l e

def __buildTable ( self, body ):
    """Build the XHTML table.

        [ body := body with a table added containing
          thumbnails and catalog data from self ]
    """

    #-- 1 --
    # [ body := body with a new table element
    #   table := that table element ]
    table = xc.Element ( body, "table", border="4",
                        cellspacing="4" )

    #-- 2 --
    # [ table := table with a new colgroup added defining
    #   the table columns ]
    colgroup = xc.Element ( table, "colgroup" )
    xc.Element ( colgroup, "col", align="center" )
    xc.Element ( colgroup, "col", align="right", valign="top" )
    xc.Element ( colgroup, "col", align="left", valign="top" )

    #-- 3 --
    # [ table := table with a thead added containing the
    #   column headings ]
    thead = xc.Element ( table, "thead" )
    headRow = xc.Element ( thead, "tr" )
    head1 = xc.Element ( headRow, "th", align="center" )
    xc.Text ( head1, "Thumbnail" )
    head2 = xc.Element ( headRow, "th", align="center" )
    xc.Text ( head2, "Size" )
    head3 = xc.Element ( headRow, "th", align="center" )
    xc.Text ( head3, "Data" )

    #-- 4 --
    # [ table := table with a tbody element added
    #   tbody := that tbody element ]
    tbody = xc.Element ( table, "tbody" )
```

Finally, we add a row to the table for each contained ArchImage object.

catweb

```
#-- 5 --
# [ tbody := tbody with rows added representing the
#   ArchImages in self, in key order ]
for archImage in self.genArchImages():
    #-- 5 body --
    # [ archImage is an archx.ArchImage ->
```

```

#      tbody := tbody with a row added representing
#              archImage ]
self.__addRow ( tbody, archImage )

```

12.7. `FormSet.__addRow()`: Generate one row of the table

This method translates one `ArchImage` instance into a table row.

catweb

```

# - - -   F o r m P h o t o S e t . _ _ a d d R o w   - - -

def __addRow ( self, tbody, archImage ):
    """Add one row to the table.

        [ (tbody is an xc.Element) and
          (archImage is an archx.ArchImage) ->
            tbody := tbody with a tr element added containing
                    the thumbnail and data from archImage ]
    """

    #-- 1 --
    # [ tbody := tbody with a new, empty tr element added
    #   tr      := that tr element ]
    tr = xc.Element ( tbody, "tr" )

```

Just to keep things uncomplicated, we'll delegate production of each of the three cells in the row to separate methods.

catweb

```

#-- 2 --
# [ tr := tr with a new td added containing the
#     thumbnail for archImage ]
self.__addThumbnail ( tr, archImage )

#-- 3 --
# [ tr := tr with a new td added containing size data
#     from archImage ]
self.__addSize ( tr, archImage )

#-- 4 --
# [ tr := tr with a new td added containing general
#     data from archImage ]
self.__addData ( tr, archImage )

```

12.8. `FormPhotoSet.__addThumbnail()`: Add the thumbnail cell

This method generates the first column of the table, containing the image thumbnail; see Section 4.3, "XHTML for the form page" (p. 7).

catweb

```

# - - -   F o r m P h o t o S e t . _ _ a d d T h u m b n a i l

def __addThumbnail ( self, tr, archImage ):
    """Add a thumbnail image cell

```

```

    [ (tr is an xc.Element) and
      (archImage is an archx.ArchImage) ->
        tr := tr with a new td added containing the
            thumbnail for archImage ]
    ""

    #-- 1 --
    # [ tr := tr with a new td element added
      #   td := that new td element ]
    td = xc.Element ( tr, "td", valign="top", align="center" )

    #-- 2 --
    # [ td := td with a new img element added whose href
      #   is the thumbnail for archImage ]
    href = "../thumb/%s.jpg" % archImage.original.catNo
    xc.Element ( td, "img", src=href, alt="thumbnail" )

```

12.9. FormPhotoSet. __addSize(): Add the size cell

This method generates the second column of the table, displaying the image size. See Section 4.3, "XHTML for the form page" (p. 7).

catweb

```

# - - -   F o r m P h o t o S e t . _ _ a d d S i z e   - - -

def __addSize ( self, tr, archImage ):
    ""Generate the image size cell of the table.

    [ (tr is an xc.Element) and
      (archImage is an archx.ArchImage) ->
        tr := tr with a new td added containing size data
            from archImage ]
    ""

    #-- 1 --
    # [ tr := tr with a new td element added
      #   td := that new td element ]
    td = xc.Element ( tr, "td", valign="top", align="right" )

```

The three lines in this cell are vertically stacked by placing each inside a `div` element.

The frame area appears only for images that have a `scan` attribute that tells the dots per inch of the scanner. If this attribute is missing, the original is from a digital camera, and we don't display the frame area at all.

To calculate the percentage frame area, we need to know the area in square millimeters of a full frame. Then we use the scanner precision, expressed in pixels per millimeter, plus the size of the image in pixels, to calculate the fraction of the frame. Here is the math:

1. The constant `FULL_FRAME_MM2` gives the area of a full frame in square millimeters: 24mm × 36mm for a 35mm film frame.
2. Since 1 inch = 25.4mm, we can convert that value to square inches by dividing by the square of 25.4.

3. To convert square inches to pixels, we multiply by the square of P_i , the dot pitch of the scanner in dots per inch. This gives us P_f , the number of pixels in a full frame at resolution P_i , which is available as `archImage.original.scan`.

For an image with x pixels, the image size as a percentage of the frame, then, is $100 \times x / P_f$.

catweb

```
#-- 2 --
# [ if archImage has a known dots/inch ->
#   td := td with a div element added, containing the
#       percentage of a full frame covered by archImage ]
if archImage.original.scan:
    fullFramePixels = ( FULL_FRAME_MM2 / (MM_PER_IN**2) *
                       (archImage.original.scan)**2 )
    thisFramePixels = archImage.wide * archImage.high
    divPercent = xc.Element ( td, "div" )
    framePercent = 100.0 * thisFramePixels / fullFramePixels
    xc.Text ( divPercent, "%.2f%%" % framePercent )
```

The width and height are added in separate `div` elements; the latter is preceded by "x". It would be nice to have the special Unicode character "x" (code point 0x00d7), but so far I haven't been able to figure out how to make it work.

catweb

```
#-- 3 --
# [ td := td with two divs added, the first containing
#   the image width from archImage, the second
#   containing 'x' and the image height ]
divW = xc.Element ( td, "div" )
xc.Text ( divW, str(archImage.wide) )
divH = xc.Element ( td, "div" )
xc.Text ( divH, "x" )
xc.Text ( divH, str(archImage.high) )
```

12.10. FormPhotoSet.__addData(): Add the cataloging data cell

This method adds all the remaining catalog data to the third column of the table row. First we add the cell's `td` element.

catweb

```
# - - -   F o r m P h o t o S e t . _ _ a d d D a t a   - - -

def __addData ( self, tr, archImage ):
    """Add the catalog data cell to the table row.

    [ (tr is an xc.Element) and
      (archImage is an archx.ArchImage) ->
      tr := tr with a new td added containing general
          data from archImage )
    """
    #-- 1 --
    # [ tr := tr with a new td element added
    #   td := that new td element ]
    td = xc.Element ( tr, "td", valign="top", align="left",
                     class_="cat-info" )
```

The elements inside this cell are stacked vertically using `div` elements. The first `div` holds the catalog number, state, and locality. The catalog number is additionally wrapped in a `span` element of class `cat-no`.

catweb

```
#-- 2 --
# [ td := td with a new div element added
#   majorDiv := that new div ]
majorDiv = xc.Element ( td, "div", class_="ident" )

#-- 3 --
# [ majorDiv := majorDiv with a new span element added
#   containing archImage.original.catNo ]
catSpan = xc.Element ( majorDiv, "span", class_="cat-no" )
xc.Text ( catSpan, archImage.original.catNo )
```

Add the state code, always present. Add the locality data (and the colon that precedes it) only if there is locality data.

catweb

```
#-- 4 --
# [ majorDiv := majorDiv with archImage's state and
#   locality added ]
xc.Text ( majorDiv, " " )
xc.Text ( majorDiv, archImage.original.state.upper() )
if archImage.original.loc:
    xc.Text ( majorDiv, ": %s" % archImage.original.loc )
```

Eventually we'll add special divs for each element such as `pose`, `beh`, and so on. For now, just put up the note content.

catweb

```
#-- 5 --
# [ td := td with a new div element added containing
#   archImage.original.note ]
if archImage.original.note:
    noteDiv = xc.Element ( td, "div" )
    xc.Text ( noteDiv, archImage.original.note )
```

13. Epilogue

These lines initiate execution of the main program.

catweb

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

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

14. Defect statistics

This program was written with Cleanroom intended functions. No self-verification or peer verification was done. Defects are counted from the first compilation. Defects are broken down into three categories: see Section 14.1, "Syntax errors" (p. 42); Section 14.2, "Strong typing errors" (p. 42); and Section 14.3, "Logic errors" (p. 44).

14.1. Syntax errors

Syntax errors caught by Python while scanning modules.

1. In function `buildWeb()`, the `for`-loop was indented one level too far, as if it were a method, not a `def`.
2. In `TaxonPhotoSet.__addData()`, this line:

```
class_="cat-info_ )
```

should have been:

```
class_="cat-info" )
```

3. In `FormPhotoSet.__addData()`, unclosed parentheses:

```
xc.Text ( noteDiv, arch.original.note
```

should be:

```
xc.Text ( noteDiv, arch.original.note )
```

14.2. Strong typing errors

These are errors that would have been caught at compile time in a more strongly typed language such as Java.

1. Somewhere on a listing I noted that I needed a line of this form:

```
import abbr as abbrModule
```

but it never got added to the imports.

2. Neglected to import the `sys` module; `catweb` needs it to retrieve its command line arguments.
3. In the `addArchive()` function, the discussion of prime 1 states that the `ArchiveIndex.readFile()` method reads the index, but the code was wrong:

```
archIndex = ArchiveIndex ( catalog, archFileName )
```

It should be:

```
archIndex = ArchiveIndex.readFile ( catalog, archFileName )
```

4. In the `addArchImage()` function, the module containing the class constructor for `BirdId` was the wrong one:

```
birdId = txnny.BirdId.parse ( txnny, rawBirdId )
```

It should be:

```
birdId = abbrModule.BirdId.parse ( txn, rawBirdId )
```

5. In `TaxonPhotoSet.__addContained()`, this code:

```
childTaxon = self.childContaining ( newTaxon )
```

should be:

```
childTaxon = self.taxon.childContaining ( newTaxon )
```

6. In `TaxonPhotoSet.__addContained()`, this code added a `Taxon` object as a value in the `self.__childMap` dictionary, but it should have been a `TaxonPhotoSet` object. The erroneous code:

```
try:
    childNode = self.__childMap [ childTaxon.txKey ]
except KeyError:
    childNode = TaxonPhotoSet ( childTaxon )
    self.__childMap [ childTaxon.txKey ] = childTaxon
```

Correct code:

```
try:
    childNode = self.__childMap [ childTaxon.txKey ]
except KeyError:
    childNode = TaxonPhotoSet ( childTaxon )
    self.__childMap [ childTaxon.txKey ] = childNode
```

7. Neglected to import the `os` module; used in `FormPhotoSet.pathName()` to assemble directory and file names into a path.

8. In `FormPhotoSet.buildPage()`, this statement has a typo:

```
titleText = ( "Shipman's bird photo index: %s" %
              self.birdId.engComma() )
```

It should be:

```
titleText = ( "Shipman's bird photo index: %s" %
              self.birdId.engComma() )
```

9. In `FormPhotoSet.__addThumbnail()`, the call to `xc.Element` omitted the name of the new element:

```
td = xc.Element ( tr, valign="top", align="center" )
```

It should be:

```
td = xc.Element ( tr, "td", valign="top", align="center" )
```

10. In `FormPhotoSet.__addThumbnail()`, the code to extract the catalog number left out a level:

```
href = "thumb/%s.jpg" % archImage.catNo
```

It should be:

```
href = "thumb/%s.jpg" % archImage.original.catNo
```

11. In `FormPhotoSet.__addSize()`, I wasn't paying attention to operator precedence. Here is the erroneous code:

```
xc.Text ( divPercent, "%.2f%%" %  
          100.0 * thisFramePixels / fullFramePixels )
```

I should have parenthesized the expression on the second line. As it was, it was grouped as `((("%.2f%%" % 100.0) * ...) * ...)`. Better code, moving the percentage calculation to a separate line:

```
framePercent = 100.0 * thisFramePixels / fullFramePixels  
xc.Text ( divPercent, "%.2f%%" % framePercent )
```

12. In `FormPhotoSet.__addData()`, this line omitted its first argument:

```
if archImage.original.loc:  
    xc.Text ( ": %s" % archImage.original.loc )
```

It should be:

```
if archImage.original.loc:  
    xc.Text ( majorDiv, ": %s" % archImage.original.loc )
```

13. In `FormPhotoSet.__addData()`, this line:

```
# [ td := td with a new div element added containing  
#     arch.original.note ]  
if arch.original.note:  
    noteDiv = xc.Element ( td, "div" )  
    xc.Text ( noteDiv, arch.original.note )
```

should be:

```
# [ td := td with a new div element added containing  
#     archImage.original.note ]  
if archImage.original.note:  
    noteDiv = xc.Element ( td, "div" )  
    xc.Text ( noteDiv, archImage.original.note )
```

14. In `TaxonPhotoSet.__buildMultiForms()`, the call to `.__buildFormLine()` omitted the first argument.

```
self.__buildFormLine ( self.__formMap[formName] )
```

It should be:

```
self.__buildFormLine ( parent, self.__formMap[formName] )
```

14.3. Logic errors

Ordinary logic errors.

1. Neglected to include the “pound-bang line” at the beginning, not to mention the comment pointing at this documentation.

- The handling of questionable identifications, such as `ab6='hamfly?'` for possible Hammond's Flycatcher, was completely dropped in design. I thought I had modified the `BirdId` class in the `abbr.py` module to handle this, but that modification never went in.
- In `TaxonPhotoSet.buildPage()`, neglected to implement the logic for the case where no forms refer to that taxon. The cases of one form and multiple forms were fine. If there is no form, then the taxon's name appears without being a link to anything.
- In `FormPhotoSet.buildPage()`, I wrote the line to create the new file when I was tired, and forgot basic Python:

```
pageFile = open ( pagePath, "new" )
```

It should be:

```
pageFile = open ( pagePath, "w" )
```

- Neglected actually to write out the index page when it is finished. Added logic:

```

#-- 3 --
# [ if index page can be created anew ->
#   index page := indexPage, serialized ]
indexFile = open ( INDEX_PAGE, "w" )
indexPage.write ( indexFile )
indexFile.close()

```

- Got the URL of the stylesheet wrong:

```
CSS_STYLESHEET = "http://www.nmt.edu/~john/scans/birds/birdindex.css"
```

It should be:

```
CSS_STYLESHEET = "http://www.nmt.edu/~john/scans/bird/birdindex.css"
```

- In `TaxonPhotoSet.buildPage`, the class of the `div` elements for higher taxa was build with the wrong separator:

```
div["class"] = "head_%s" % self.taxon.rank.code
```

It should be:

```
div["class"] = "head-%s" % self.taxon.rank.code
```

- In `FormPhotoSet.__addThumbnail()`, the logic that sets up the image link is brain-dead:

```
href = "thumb/%s.jpg" % archImage.original.catNo
xc.Element ( td, "img", src="href", alt="thumbnail" )
```

The second line should be:

```
xc.Element ( td, "img", src=href, alt="thumbnail" )
```

- In `FormPhotoSet.__addThumbnail()`, the path to the thumbnail has to go up one level before it goes back down to the thumbnail directory:

```
href = "thumb/%s.jpg" % archImage.original.catNo
```

It should be:

```
href = "../thumb/%s.jpg" % archImage.original.catNo
```

10. In `TaxonPhotoSet.__buildMultiForms()`, neglected to place the taxon's name in the higher-taxon `div`. New first prime:

```
#-- 1 --  
# [ parent := parent with self's name added as text ]  
self.__webName ( parent )
```

11. In `FormPhotoSet.pathName()`, forgot that bird codes can be right-blank padded. The cure is to call `.rstrip()` on all bird codes before forming them into pathnames.
12. Neglected to implement the design goal of suppressing a genus name headings if no form is referred to it. Rewrote `TaxonPhotoSet.buildPage()`: don't allocate a `div` until we have eliminated this case.
13. `FormPhotoSet.pathName()` generated the same path name for questionable forms as for regular ones. The fix is to append "-q" for questionable forms.