The Infrastructure of the Bay Area Wildflower Guide

Revision 0.1 (2022-07-23)

The Bay Area Wildflower Guide (BAWG) is a website generated by a Python script from human-readable asset files. The term "BAWG" refers equally to the website, to the script that produces it, and to the entirety of files that go into the guide.

This document currently describes the asset files in enough detail to hopefully allow a different guide to be produced. As part of that, it describes some aspects of the Python script at a high level to facilitate understanding of how the assets are transformed into the website files, but it concentrates mainly on format of the information in the asset files themselves.

Although I am gradually making the scripts more generic and moving more information into the asset files, some aspects of the generated website are still hardcoded into the scripts, including the name and author of the guide! Until I fix that, you'll have to do a little script editing to fix the generated website to your liking.

Table of Contents

Introduction

I began the BAWG as a very simple script to organize my text notes about how to distinguish certain related species of flowers. As such, the script included an ability to generate freeform taxonomic hierarchies, e.g. clarkia is a genus of flowers, and it includes a few local species. To provide easy access to example photos of each species, the script would reformat the text notes into HTML and include image links for any photos that had the same base filename as a text file.

Over time, I made the script much smarter about keeping track of the underlying scientific taxonomy, partly as a way to find mis-sorted species, and partly as a way to apply different properties to different parts of the taxonomy hierarchy. Along the way, I added features to gather additional information from other sources, especially from observations I'd submitted to the iNaturalist website. I also generated additional helpful HTML, including summaries of flowers organized by color and links to my own glossaries of terms.

The BAWG began as a wildflower guide, and it still has the best support for flowers, but I've extended it to include other bay area wildlife, including non-flowering plants, animals, etc. It is by no means a complete description of every form of life found in the bay area. Instead, I've concentrated my efforts on those species that I've photographed (which roughly corresponds with the most common local species). I've also included notes on similar species that I haven't photographed so that I can more quickly identify anything that I encounter in the future.

Each species (or occasionally, each subspecies and/or variety) has its own HTML page. In addition, groups of related species are linked from a higher-level "parent" page, and the lower-level species are the "children" of that parent. An entire hierarchy of parent-child relationships is built so that a species can be traced all the way back to the root of the tree of life, with a page for each "taxon" group along the way.

This "tree of life" analogy makes its way into the colors of links in the BAWG. A "leaf" taxon is one which has no children, and any link to it from another page is colored green. A "branch" taxon is one which has one or more children, and any link to it from another page is colored brown. (Although actually there is a third color, black, which is used for leaf pages that have no photographs. In this case, the link color lets the user know that there is little to be gained by following the link.)

Although the BAWG script supports reading information from quite a few different types of files, all of these files are optional. You can begin as I did with just a few txt files and/or photos, then add any auxiliary files you want as you need them.

If you want to start a new guide from scratch, you can either delete all asset files and generated website files listed below, or run the script with the -dir option. Script options are described in page 13.

Summary of Files

Documentation

```
README.md                   # Readme for GitHub
bawg notes.txt              # A random collection of my notes and to-do items
docs/BAWG.odt               # This file
docs/*.txt                  # Earlier documentation that I plan to port into this document
```

Scripts and related source files

```
src/*.py                    # Python script files
src/*.js                    # Javascript that supports the website UI
src/bawg.css                # CSS for the website
```

Asset files

```
txt/*.txt                   # Describe the contents and organization of web pages
photos/*.jpg                # Photographs of taxons
glossary/*.txt              # Definitions of terms for various subsets of the taxon hierarchy
figures/*.svg               # Photos with labels added to help illustrate glossary terms
figures/*.jpg
data/observations.csv       # Information about observed taxons exported from iNaturalist
data/parks.yaml             # Mapping of place names in observations.csv to preferred names
data/ignore_species.yaml    # List of taxons to ignore from observations.csv
data/taxon_names.yaml       # Mapping between common and sci. names for taxons without txt files
jepson_glossary.txt         # List of terms that we want to link to the Jepson glossary
other/*.txt                 # Text for the body of each top-level HTML file
other/footer.html           # HTML for a standard page footer
```

Generated website files

```
html/*.html                 # Web pages for each taxon
thumbs/*.jpg                # Thumbnails for all photos in photos/*.jpg
*.js                        # Copies of src/*.js with comments removed to save bandwidth
pages.js                    # List of all taxon pages for use by the Javascript search bar
photos.js                   # List of all photos and figures used on taxon and glossary pages
url_data.json               # Hash for each taxon page used by the Javascript offline cache
bawg.css                    # Copy of src/bawg.css with comments removed to save bandwidth
*.html                      # Copy of other/*.txt with headers, footers, and formatting added
```

Static website files

```
manifest.webmanifest        # Required for PWA (progressive web app) support
icons/*                     # Standard icons used on many guide pages
favicon/*                   # Icons and related information for use by the browser and desktop
```

Git helper files

```
.gitattributes
.gitignore
```

Where Pages Come From

A file in txt/*.txt is called a “txt file”, and its contents are the “txt”. Every txt file creates a corresponding HTML file with the same base filename.

Txt can also specify the names of child pages along with other information about each child. Even if there is no other source of information for a child pages, the script still creates a page for that child with its name and position in the hierarchy. I’ve particularly taken advantage of this for genus pages, which specify enough information about their member species that separate txt files for the species are usually unnecessary.

Photos are normally associated with a txt file or with a child named in the txt. If a photo’s filename doesn’t match any known taxon, it creates its own page for the photo, but in my current workflow this is always a mistake, and the created page generally triggers an error because it can’t find its place in the hierarchy.

The script imports my iNaturalist observations, which also includes many of the taxonomic ranks that each observed taxon is a part of. This taxonomic hierarchy is not fully translated into pages since there are more levels of detail there than I generally care about, but I’ve selected certain levels of hierarchy to automatically turn into pages if they don’t otherwise exist. The hierarchical relationships are also used to fill in any parent-child relationship that might otherwise be missing.

These additional ways of creating pages mean that although I currently have about 500 txt files, over 2000 HTML pages end up being generated.

Tiny Example

This section demonstrates how a very simple guide can be constructed containing a single genus that contains two species. Note that many of the example filenames include spaces. I’m a rebel that way.

txt/example genus.txt

```
sci:genus Examplus

Here is some introductory text.

==red-headed example:Examplus rotus
. recognized by its red color

==blond example:Examplus gelbus
. recognized by its yellow color
```

The name of the txt file, “example genus.txt” indicates the name of the resulting HTML file “example genus.html” and also implies the common name of the taxon, “example genus”. Note that most modern operating systems allow spaces in filenames, which is very useful for this purpose.

The txt begins with a declaration of the scientific name of the taxon, “genus Examplus”. (Pardon my terrible faux-Latin.)

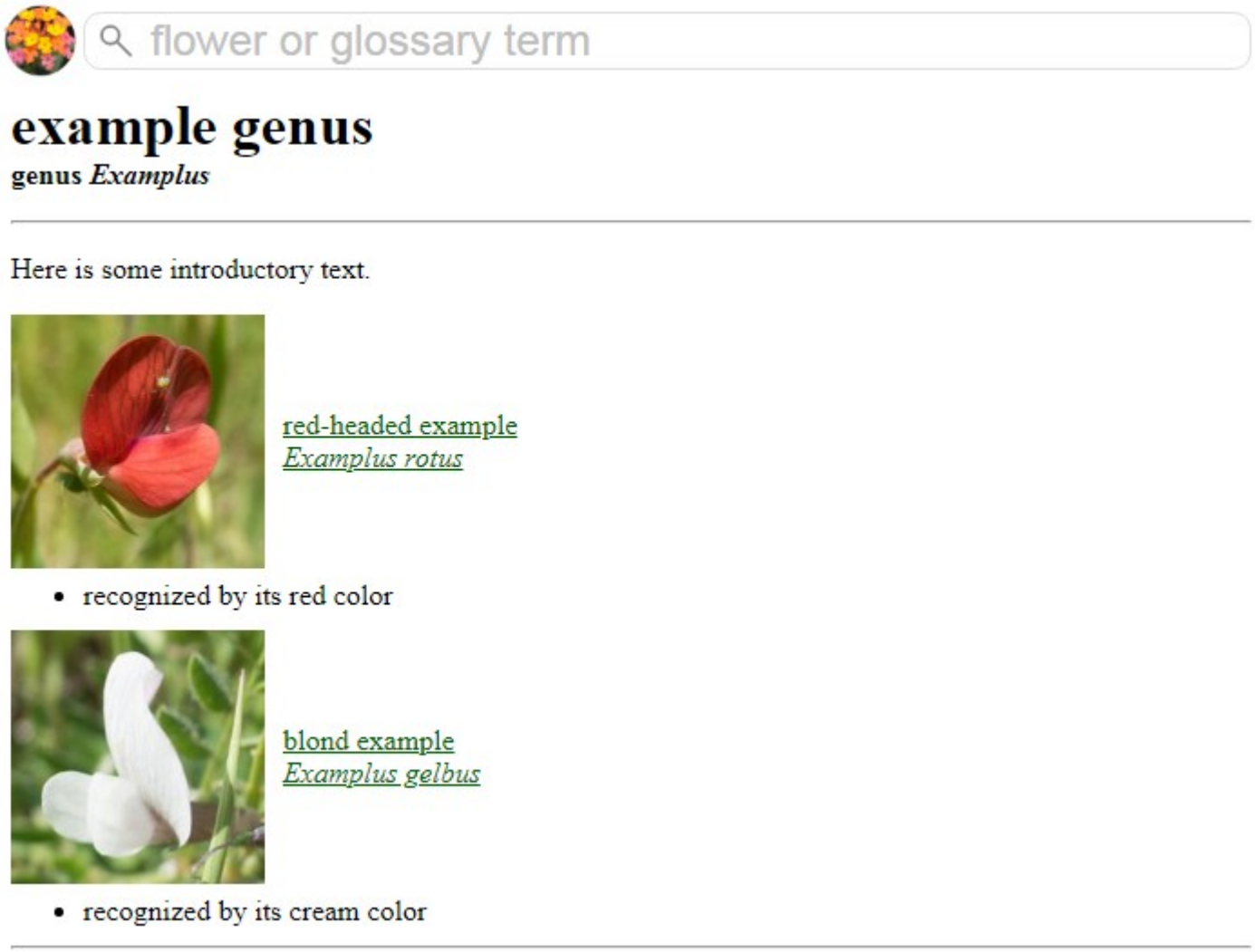
The txt continues with declarations for the two species under the example genus. Each child is declared with both a common name, e.g. “red-headed example” and a scientific name, e.g. “Examplus rotus”. Each child is followed by information that is useful when distinguishing among species within a genus.

Finally, I supply photos of each species:

```
photos/red-headed example.jpg
photos/blond example.jpg
```

The result is three HTML pages: one for the genus and one for each species.

html/example genus.html



Every taxon page has a standard header with an icon that links to the guide's home page when clicked. For this tiny example with only the listed files, this is a broken link since there is no index.html file. Next to the home icon is a search bar. It can be used to quickly navigate by name to any of the three taxons in this example.

Below the header are the names of the taxon, common name followed by scientific name. Below that is the remaining text from the txt file. Where the txt file links to each child taxon (with the '==' prefix), the HTML file shows a thumbnail photo for each child page and links the child by both common name and scientific name. The distinguishing inforomation is also listed for each child.

html/red-headed example.html

The child pages are similar, with the addition of a link back to their parent. Note that in the absence of txt specifically for the child, the distinguishing information from the parent page is copied to the child page.

On these child pages, rather than linking the photo thumbnail to another page, the thumbnail links to the original full-size version of the photo.

There are many more elements that can be added to a page, more formatting options, and additional ways to create and link page. All of these options are described in this document.

Complete Example

The below example is from the full BAWG.

html/shooting stars.html

flower or glossary term

shooting stars

genus *Primula*
Member of primrose family (family *Primulaceae*)
flowering plants (subphylum *Angiospermae*)

There are no other wild species of this genus in the bay area.

henderson's shooting star
Primula hendersonii

- peduncle purple (or mix of purple/green)
- leaf blade length generally < 2 × width

padre's shooting star
Primula clevelandii

- peduncle bright green
- leaf blade length generally > 2 × width

Chris's observations: 46 (42 are research grade) [show details]

Taxon info: iNaturalist – CalFlora – Jepson eFlora

Bay Area species: iNaturalist – CalFlora

Bay Area Wildflower Guide © Chris Nelson Contact me

This page has more information than the tiny example above.

Near the top, the page lists what interesting higher-level taxon the shooting stars are a member of. It also includes a note indicating that the taxon is complete.

Near the bottom, the page lists how many observations I've made of the genus. It also includes helpful links to other websites with information about the taxon.

Finally, there is a standard footer with a link back to the guide's home page and to my contact information.

All of this additional information is configured with a combination of txt commands and with additional data files input to the script. These are covered in the remainder of this document.

Browser Support

The BAWG is written to look decent on a range of devices, from large desktop to a small phone screen. Text is sized according to the browser defaults (which the user may have adjusted to her preferences). Photo thumbnails have a fixed size on larger screens, but shrink on smaller devices to avoid crowding the text.

JavaScript Functions (or lack thereof)

The BAWG uses JavaScript for various functions, but I've tried to make it degrade as gracefully as possible if JavaScript is disabled.

Search

The BAWG uses only a static web server without intelligence, so the search box is handled entirely within the web page using JavaScript in search.js. A generated JavaScript file, pages.js, lists all supported search terms and the page where each can be found.

If JavaScript is disabled, the search box is grayed out and displays a message that "search requires JavaScript". I've placed manual links in the top-level index page to allow the user to traverse to the top of the taxon hierarchy and from there to any desired taxon.

Photo Gallery

By default, links to photos and figures in the BAWG simply link directly to the JPG or SVG file and let the browser display the image as it sees fit. However, the user sees these links only if JavaScript is disabled.

If JavaScript is enabled, the script replaces each photo/figure link with a link to the photo gallery page (gallery.html) with a URL extension specifying the image being linked. The photo gallery page relies on JavaScript in gallery.js to load the specified image. The user can then use the mouse or fingers to pan and zoom

the photo. If the image is listed in another generated JavaScript file, photos.js, the photo gallery also lets the user browse through all photos on the same page.

The internet is inconsistent about whether the browser's "back" function closes the gallery or whether instead clicking somewhere off the photo closes the gallery. To meet the user's expectations in all cases, the user can return from the photo gallery to the originating page using either method.

Offline Caching

The BAWG is built as a progressive web app (PWA) that can be cached for offline use. This is useful if the user wants to consult the BAWG when outside of data range (e.g. on a hike away from cell towers) or when data is cheaper at home than away from home (e.g. Wi-Fi vs. cellular connection).

Since the required data storage is substantial, files are cached for offline use only when the user presses the shiny green button on the home page. Once the data is cached, the only online data transfer is an occasional check to see if an update is available. The user also determines if/when she wants to perform the update, so there is no danger of accidentally consuming a lot of metered data.

Since a partial update could leave the BAWG with broken links or images, the caching scripts continue to use the old files until all updated files are cached. Only then does it switch to the updated database, and it discards any old files that are no longer used.

If JavaScript is disabled, the buttons to control offline caching are gray and disabled.

Literals and Variables

In this document, literal text that should be copied exactly is highlighted like this: literal text.

Text that should be modified to fit the scenario is called variable text, and it is highlighted like this: *variable text*. In this case, the highlighted text describes what should go in that position.

For example, a txt file can specify the common name with com: *name*. Therefore, when you want to specify the common name in a txt file, you should copy com: exactly, and put the desired name in place of *name*. E.g. “com: california poppy”.

Running the Script

The script can be run from anywhere, but I generally assume that you're running it from the root directory of the BAWG respository. In the absence of command-line options, the script reads and writes files in the same repository area as the script. E.g.

```
% src/bawg.py
No files modified.
```

If any HTML pages are modified, the script generates _mod.html with links to all of the modified pages and loads it into a browser. In this way, you can quickly browse the results of changes you make to the script inputs.

Script Performance

On my Windows desktop with a solid-state drive (SSD), the script runs in 6–7 seconds on the full BAWG, assuming that no more than a few photo thumbnails need to be regenerated.

In normal operation, the vast majority of the script time is spent writing to disk the more than 2000 HTML files. If you run anti-virus software (such as Windows Security Essentials) that checks every new file for malware, this can slow the script significantly (about 4x for me). For this reason, I recommend setting your anti-virus options to exclude the BAWG directory from virus checking if you can.

Thumbnail Conversion

The BAWG script generates a thumbnail for each JPEG in the photos directory. For speed, the script doesn't bother to regenerate a thumbnail if it already exists and is younger than the associated full-sized photo. Thus, no more than a few thumbnails usually get generated for any script run. For reference, regenerating all 2000+ thumbnails for the BAWG takes 1 or 2 minutes on my system.

The script needs the assistance of an external program to resize photos into thumbnails. It can use either ImageMagick or IrfanView, but it prefers ImageMagick for its smaller thumbnail file sizes.

The script searches your path for these programs. The program name can be magick (used by ImageMagick 7), mogrify (ImageMagick 6), i_view32 (32-bit IrfanView), or i_view64 (64-bit IrfanView). On Windows it also searches in the Program Files and Program Files (x86) directories for an installation directory with a name starting with ImageMagick or IrfanView. (The path search should also work under Unix/Linux, although I have not tested it.)

If neither of these programs can be found, then thumbnails are not generated, and the HTML tells the browser to fetch and shrink the full-sized photos instead. However, you should try to install one of the above programs in order to generate a more bandwidth-friendly website.

Script Command-Line Options

The BAWG script has reasonable default behavior when no command-line options are given. The options below, therefore, are primary useful to help debug failures.

-dir

The -dir *name* argument tells the script the directory in which it should be run. The *name* parameter can be an absolute path or relative to the current directory. The named directory must already exist. All output files are written to this directory, and most input files are read from this directory, with the following exceptions:

```
src/*.py            # All Python scripts are read from the standard BAWG directory.
```

src/*.js # Javascript files are copied from the standard BAWG directory.
src/bawg.css # The site CSS file is copied from the standard BAWG directory.
icons/* # If not found in the named directory, icons are copied from the BAWG directory.
I use the -dir *name* argument especially for small experimental debug runs. You might choose to use it to start a fresh guide rather than overwriting the standard BAWG files.

-steps

The -steps argument tells the script to describe each major step of processing that it performs and the various data files that it reads (or skips). E.g.

```
% src/bawg.py -steps
Reading other/footer.html
Reading data/parks.yaml
Step 1: Reading primary names from txt files
Step 2: Parse child info
Step 3: Attach photos to pages
Step 4: Parse taxon_names.yaml
Reading data/taxon_names.yaml
Step 5: Update taxonomic links
Step 6: Add explicit and implied ancestors
Step 7: Create taxonomic chains from observations.csv
Reading taxon hierarchy from data/observations.csv
Step 8: Assign ancestors to pages that don't have scientific names
Step 9: Assign default ancestor to floating trees
Step 10: Propagate properties
Step 11: Apply create and link properties
Step 12: Read observation counts and common names from observations.csv
Reading data/ignore_species.yaml
Reading observation data from data/observations.csv
Step 13: Apply remaining properties
Step 14: Parse remaining text, including glossary terms
Reading data/jepson_glossary.txt
Writing HTML
```

The step numbers are useful for the -tree argument below. But note that the step numbers may change occasionally without this document being updated.

-tree

The -tree *n* argument tells the script to emit its internal hierarchy of pages after step *n*. (See the -steps argument above.) E.g.

```
% src/bawg.py -tree 2
example genus (genus Examplus)
  *red-headed example (Examplus rotus)
  *blond example (Examplus gelbus)
```

Each child page is indented below its parent. If the child is a shadow page (page 23), its name is enclosed in [brackets]. The child's name is also prefixed with a symbol indicating the type of parent-child relationship: a taxonomic child only (-), a real child only (+), or both a real and a taxonomic child (*).

Note that depending on your heirarchy, there may be multiple tops of trees. In addition, a page may be the child of more than one parent. In this case, the child page is marked as a {repeat}, and any duplicate hierarchy below that child is omitted.

The -tree *n* argument can be followed by an optional *taxon*. In this case, instead of writing out all trees after step *n*, it writes out only the tree starting from *taxon*.

The -tree *n* argument can also be followed by an optional keyword props. This tells the script to also write all of the rank properties for each page. If props is used in combination with *taxon*, they can be used in either order following -tree *n*.

```
% src/bawg.py -tree 13 shooting stars props
shooting stars (genus Primula)
  do casual_obs_promotion
  warn color_requires_photo
  caution default_completeness_genus
  do link_bayarea_calflora
  do link_bayarea_inaturalist
  do link_calflora
  do link_inaturalist
  do link_jepson
  do obs_fill_alt_com
  error obs_fill_com
  error obs_fill_sci
  warn obs_requires_photo
  warn photo_requires_color
```

```
  *henderson's shooting star (Primula hendersonii)
    do casual_obs_promotion
    warn color_requires_photo
    caution default_completeness_genus
    do link_bayarea_calflora
    do link_bayarea_inaturalist
    do link_calflora
    do link_calphotos
    do link_inaturalist
    do link_jepson
    do obs_fill_alt_com
    error obs_fill_com
    error obs_fill_sci
    do obs_promotion
    warn obs_promotion_above_peers
    warn obs_requires_photo
    warn one_child
    do outside_obs
    do outside_obs_promotion
    warn photo_requires_color
  *padre's shooting star (Primula clevelandii)
    do casual_obs_promotion
    warn color_requires_photo
    caution default_completeness_genus
    do link_bayarea_calflora
    do link_bayarea_inaturalist
    do link_calflora
    do link_calphotos
    do link_inaturalist
    do link_jepson
    do obs_fill_alt_com
    error obs_fill_com
    error obs_fill_sci
    do obs_promotion
    warn obs_promotion_above_peers
    warn obs_requires_photo
    warn one_child
    do outside_obs
    do outside_obs_promotion
    warn photo_requires_color
```

-no_error_limit

To avoid burying you in errors, the BAWG script exits after 10 errors are found by default. The -no_error_limit argument tells the script to continue to the end regardless of how many errors are encountered. In either case, any errors prevent the new HTML from being written out, and the script exits with a non-zero return code.

-incomplete_keys

The -incomplete_keys argument tells the script to write the names of the five most observed genuses that don’t have a complete key. E.g.

```
% src/bawg.py -incomplete_keys
acorn woodpecker (Melanerpes formicivorus)
manzanitas (genus Arctostaphylos)
bumble bees (genus Bombus)
mallard (Anas platyrhynchos)
gilias (genus Gilia)
```

This is a rather specialized argument that I added for a specific purpose while writing the BAWG and never updated. You may want to modify the script if you have a similar (but not identical) purpose in mind.

-jepson_usage

The -jepson_usage argument tells the script to write the Jepson glossary terms that are most often linked from the guide. I find this handy when considering what additional terms I should add to my own glossaries.

-debug_js

The -jepson_usage argument tells the script to not remove console-logging function calls when copying the Javascript sources to the area used by the HTML. When the Javascript is working normally, these functions just bloat the Javascript size and potentially confuse the user, so I find it best to make releases with these calls removed. However, they are obviously useful when debugging my Javascript. Hence, the script argument.

The Javascript console-logging functions are as follows:

```
console.error()
console.warn()
console.info()
console.log()
```

-without_cache

The -without_cache argument tells the script to replace sw.js and swi.js with a no_sw.js, a stub script that prevents the use of the offline web cache. Normally the user can decide whether to use the offline cache and when to update or delete files from the cache. However, a bad bug in the Javascript might prevent the user from properly interacting with the cache. The replacement no_sw.js script might help rescue us from that situation.

I added this argument when developing the offline-cache code. Hopefully none of us will ever need it in a released environment, however.

Names

Each page generated by the BAWG must have a common name, a scientific name, or both. These names are listed in the HTML, of course, and the user can search for a name via the Javascript. In addition, the BAWG script uses the names to correlate and cross-link data sources in order to provide additional information for each page that may not be in the txt files.

Scientific Name

A scientific name is a name used by scientists to uniquely identify a taxon.

Species use the Latin-based binomial nomenclature popularized by Carl Linnaeus, e.g. *Eschscholzia californica*. In order to recognize the name as a scientific name, the script expects the first word (the genus name) to be capitalized and the second word (the species epithet) to be lowercase. The script automatically applies the proper italicization when writing the HTML

A hybrid is handled in the same way as a species (and is treated as the same level in the taxonomic hierarchy). In the txt and yaml files, a hybrid is indicated by prepending the species epithet with a capital X, e.g. "Iris Xgermanica". The script recognizes this notation is indicating a hybrid, and it writes the name to HTML using the × symbol, e.g. *Iris* × *germanica*.

Except for the special symbol for hybrids, I have never seen a scientific name that includes non-ASCII Unicode characters. Theoretically the script should be fine if they do occur, but it hasn't been tested in any real way.

A subspecies is designated with the species name followed by "ssp." and then the subspecies name, e.g. *Eschscholzia californica* ssp. *californica*.

A variety is designated with the species name followed by "var." and then the variety name, e.g. *Amsinckia menziesii* var. *intermedia*.

A taxon at another level (e.g. genus or above) is designated using its taxonomic rank followed by its taxon name, e.g. "genus *Eschscholzia*". In order to recognize the name and rank, the script expects the first word (the rank) to be lowercase and the second word (the taxon name) to be uppercase. The following ranks are currently supported as part of the scientific name, although the list can easily be extended in rank.py:

- kingdom
- phylum
- subphylum
- superclass
- class
- subclass
- superorder

- order
- suborder
- infraorder
- parvorder
- superfamily
- epifamily
- family
- subfamily
- supertribe
- tribe
- subtribe
- genus
- subgenus
- complex

A genus can also be written as the genus name followed by “spp.” The script converts this to its prefered form for output. E.g. “Eschscholzia spp.” is converted to “genus *Eschscholzia*”.

Although a taxon is expected to have exactly one scientific name, sources may disagree as to what that name should be. The script has a limited ability to track additional scientific names used by other sources (page 54).

Elaborated vs. Stripped Scientific Name

The script actually consideres the above examples to be “elaborated” scientific names. A “stripped” scientific name removes the following elaborations:

- The rank of a higher-level taxon (leaving a single-word name).
- The “X” of a hybrid (leaving a two-word name).
- The “ssp.” of a subspecies (leaving a three-word name).
- The “var.” of a variety (leaving a three-word name).

If a stripped name is supplied that doesn’t match a known elaboration, the script assumes that a single-word name corresponds to a genus, and a three-word name corresponds to a subspecies.

The same stripped name may occasionally be used to refer to taxons at different ranks. An elaborated scientific name is guaranteed to refer to a single taxon.

Stripped names are supported because iNaturalist always uses stripped scientific names. However, the script prefers that you use elaborated names to avoid ambiguity.

No Scientific Name

If a page has a common name, it does not need to have a scientific name. If a scientific name is not explicitly assigned, it may instead be implicitly assigned if the common name finds a match (e.g. in the iNaturalist

observations). To prevent this, the scientific name can be set to n/a. This does not actually assign a scientific name, but instead disclaims a scientific name, preventing one from being implicitly assigned.

The n/a declaration also satisfies the no_sci property (page 47), which would otherwise complain if the page doesn't have a scientific name.

Common Name

A common name is the name used by most people to colloquially refer to a taxon, e.g. "california poppy".

Because the script was built to my personal preferences, common names are expected to be entirely lowercase. If a name is used that could be a common name or scientific name, the script relies in the case to distinguish the two types of name. Where the context is unambiguous, a common name can potentially use uppercase letters, but be aware that the script is much less tested for that case.

Unicode characters are supported in the common name.

The common name generally depends on the local language, but even within a language different people may use different names in different regions or even within a region. However, the BAWG expects to be used at a local scale, so it generally supports one common name which is typically the most common local name. The BAWG has a limited ability to track additional common names used by other sources.

Multiple pages can use the same common name as long as each also has a scientific name to uniquely identify it. If the script tries to find a page by common name alone, and multiple pages share that name, it matches the one that corresponds to a txt file with the same filename. If there is no such page, the look-up fails on the ambiguous name.

No Common Name

If a page has a scientific name, it does not need to have a common name. If a common name is not explicitly assigned, it may instead be implicitly assigned if the scientific name finds a match (e.g. in the iNaturalist observations). To prevent this, the common name can be set to n/a. This does not actually assign a common name, but instead disclaims a common name, preventing one from being implicitly assigned.

The n/a declaration also satisfies the no_com property (page 47), which would otherwise complain if the page doesn't have a common name.

File Name

When the script writes an HTML file, it chooses the HTML filename from one of the following choices, with preference going to the first choice that fits:

- the same as the original txt file name (but with the .html extension).
- the common name (if it is unique).
- the unelaborated scientific name (if it is unique).
- the elaborated scientific name (which must be unique).

If a name includes any Unicode characters, they are converted to simple ASCII for the filename.

Sources of Names

A page can get its common name and/or scientific name from a variety of sources.

A page can infer its common name or scientific name from its filename, based on the filename's capitalization.

A txt file can specify an explicit common name using the 'com' attribute and/or it can specify a scientific name using the 'sci' attribute (page 32). This overrides any inferred name.

When a txt file declares its children, it can specify a common name and/or scientific name for each child (page 33).

A txt file can declare any number of ancestors with a common name and/or scientific name (page 52).

Correspondences between common and scientific names are specified in taxon_names.yaml (page 42).

Page Hierarchy

A page is created for each text file.

Among other information, each txt file can also specify one or more children of the page, and pages are also created for those children. Each child may have its own txt file to specify additional details about itself and perhaps specify additional children of its own.

These parent-child relationships form a hierarchy with any number of ancestors and descendents. (You could also think of it as a family tree.)

A typical hierarchy might include a family page with genus children, where each genus has species children, and it may continue down to subspecies/varieties. However more or fewer levels of hierarchy can be specified as desired, including groupings that don't correspond to scientific taxons. The various pages may all share a single top-level ancestor to link them all together, or there can be multiple disjoint hierarchies.

The term “descendent” refers to a child, a child of a child, etc.

The term “ancestor” refers to a parent, a parent of a parent, etc.

The page hierarchy can be constructed however desired as long it contains no circular loops in which a page is its own ancestor or descendent.

A page can have any number of children and can also have any number of parents (be the child of multiple other pages). However, in most cases a typical page has only one parent. If a page has multiple parents, they may share a common ancestor or not.

Rank-Based Taxonomy

The script does its best to fit the explicit page hierarchy into a rank-based taxonomy. The more complete this mapping is, the more information the script can correlate addiitional information and properties for each page. On the other hand, if the page hierarchy violates the assumptions of taxonomy (e.g. by having a genus be under a species), the script will fail with an error.

The script can infer some aspects of taxonomy directly from scientific names. E.g. *Eschscholzia californica* is known to be a species under genus *Eschscholzia*. The txt files can also specify additional taxonomic information, and the script can also read taxon relationships from the iNaturalist data.

For example, a large portion of the BAWG’s flower hierarchy is built as follows:

- A txt file is created for each genus to specify the local species (as child pages) and the distinguishing features of each.

- Photos are created for each species, but no txt file is needed for most species since the species info is specified by the genus's txt file.
- Information from iNaturalist is used to build a taxonomic hierarchy linking the various genuses, e.g. by automatically creating a page for each family and a top level page linking all families.

Shadow Pages vs. Real Pages

From its various information sources, the script can build a more complete taxonomy than you necessarily want to record to HTML pages. For this reason, the script infers many pages initially as "shadow pages". Shadow pages help the script to organize the taxonomy, but they aren't written out as HTML pages.

You can, however, choose to promote selected shadow pages into real pages that are then output to HTML. For example, you can promote to real pages all shadow pages at the "family" rank under "flowering plants".

Shadow Links vs. Real Links

The script also infers shadow links between taxonomically related pages, whether those pages are shadow or real. These shadow links aren't output to the HTML unless they are first promoted to real links.

Sources of Pages

A real page can come from any of the following sources:

- A txt file describes a page (page 27).
- A txt file declares a child page (page 33).
- A txt file declares a subset page.
- A photo implies a page (page 24).
- A create property promotes a shadow page.

A shadow page can come from any of the following sources:

- An attribute declares an ancestor page (page 52).
- An iNaturalist observation has information about the taxon (page 37).
- taxon_names.yaml names a page (page 42).

Photos

Photos of taxons should be in the photos directory with a .jpg extension.

The photo should use as its filename either the common or scientific name of its taxon. E.g. “photos/california poppy.jpg” is a photo of a california poppy.

Photo Suffix

In order to have multiple photos of a taxon, each photo must have a suffix separated from the name by a comma. E.g. “photos/california poppy,1.jpg” is a photo of a california poppy, and additional poppy photos can be put in the photos directory with different suffixes.

The suffix must start with a digit (0 – 9) or a hyphen (-). This requirement helps distinguish the suffix, since a taxon’s common name can also include a comma. The remainder of the suffix can include any characters allowed in a filename as long as it doesn’t include a comma (,) or period (.).

Photos on a taxon page are ordered **alphabetically** by their suffix names. Note that an alphabetical sort of numerical suffixes might not work as expected, e.g. “,10” sorts before “,2”.

Thumbnail and Full-Size Photos

Photos associated with a taxon are shown as a row of thumbnails near the top of its web page. The script automatically generates thumbnails for all photos. Since the thumbnail conversion process is slow, the script compares the timestamps on the photo and thumbnail files and only updates the thumbnails when necessary.

The user can click on a thumbnail to navigate to the full-size photo in a photo gallery (page 10).

Representative Photo

By default, the alphabetically first photo represents the taxon when linked from a parent page. Or the parent can choose a photo with a different suffix to represent the child (page 35).

If a page doesn’t have an explicit photo for its taxon, it is represented in links by the representative photo of one of its children. By default it uses the first listed child that has a photo, but a different child can be selected (page 36).

Photo from CalPhotos/Calflora

Instead of or in addition to local photographs, the BAWG can also link to photos hosted on CalPhotos or Calflora. In this case, a placeholder is used instead of a regular thumbnail, and the placeholder is linked to the CalPhotos or Calflora photo page. The format of the link is described on page 51.

Figures

Illustrative figures should be in the figures directory with a .svg or .jpg extension. A figure can have any legal filename.

A figure can easily be displayed and/or linked from a txt file (pages 29 and 30).

Since figures appear on only a few pages, the script doesn't bother to make thumbnails for them at this time.

SVG Hints

A full-size SVG is displayed in the same manner as a full-size photo (page 24), allowing the browser to handle the details. However, SVG can specify an absolute display size, which prevents the browser from displaying the figure in an optimal size for the user's device. To prevent this, ensure that the SVG is written in a way that scales automatically.

In Inkscape (as of version 0.92), the following procedure will ensure that the SVG scales automatically:

- Edit -> XML Editor...
  - Select the top element, "svg:svg ..."
    - Select "height"
      - Input the value "100%" and press ctrl-enter
    - Select "width"
      - Input the value "100%" and press ctrl-enter
- File -> Document Properties...
  - In the box for "Scale", open the section for "Viewbox..."
    - Set X to 0
    - Set Y to 0

Although it's tempting to edit the SVG directly, doing so will cause it to be drawn to the wrong scale relative to the viewbox. Editing the XML properties within Inkscape will cause Inkscape to rescale the drawing correctly.

Txt Files

Txt files provide the majority of the input to a guide. They describe the taxons of interest and the relationship betweeen taxons. They also provide information for making links to external websites, and they declare properties used to generate additional information and to check for mistakes.

“Txt” is of course an abbreviation of “text”. In the BAWG, txt refers specifically to the contents of the files in the txt directory.

Txt files are in UTF-8 format.

Txt Formatting

Txt files are secretly HTML files with a lot of script assistance. If the below formatting aids don’t support the formatting that you’d like, feel free to code the necessary HTML directly into your txt file.

Line and Paragraph Formatting

Paragraphs

Regular text on consecutive lines are treated as a single paragraph, and a blank line separates paragraphs. The script surrounds each paragraph of text with the HTML and tags. For example:

```
This is
paragraph 1.

And this is paragraph 2.
```

The above txt results in the following browser formatting:

This is paragraph 1.

And this is paragraph 2.

Bulleted Lists

A bulleted list can be created by putting one or more periods (.) at the beginning of each line. The number of periods indicates the depth that the line should be indented within the list. For example:

```
The following is a bulleted list.
. Item 1.
.. Item 1a.
.. Item 1b.
. Item 2.
. Item 3.
.. Item 3a.
```

The above txt results in the following browser formatting:

The following is a bulleted list.

- Item 1.
  - Item 1a.
  - Item 1b.
- Item 2.
- Item 3.
  - Item 3a.

Unlike normal paragraphs described above, an item in a bulleted list cannot have a newline within it. A new line that doesn't begin with a period starts a new paragraph.

A list can be used within key text (page 34), and the key text also includes any immediately adjacent non-list text paragraph before or after the list.

Boxes

If a taxon has a lot of child taxons or other information in it, it can be useful to visually group portions of the information together with boxes.

A boxed section is started by putting an left square bracket ([) on a line by itself. The section is closed by putting a right square bracket (]) on another line by itself. Boxed sections can be nested arbitrarily deep. For example:

```
This text isn't in a box.
[
This text is in a box.
[
And this text is in <b><i>two</i></b> boxes.
]
Boxes are a good way to group parts of the page together, e.g. similar species within a genus.
]
```

The above txt results in the following browser formatting. (This may be slightly confusing since this document puts the screenshot from the browser in a box, so keep in mind that the outer box is not actually displayed within the browser.)

Embedded Figures

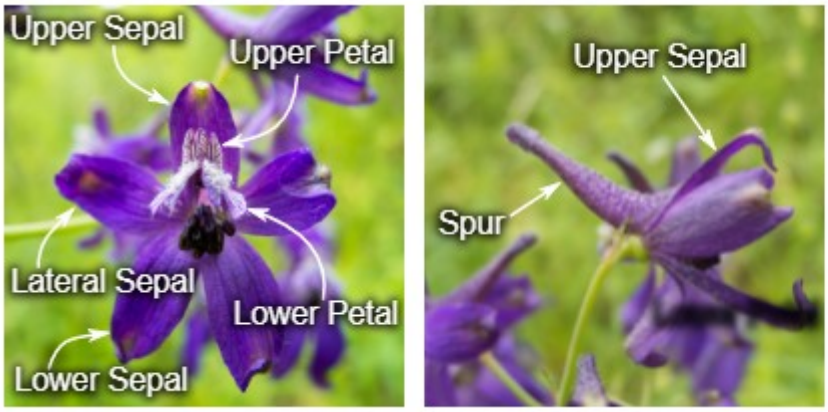
A thumbnail image of a figure can easily be embedded in the text that automatically links to the full-sized figure. Figure images should be in the figures directory with either the extension .svg or .jpg.

Include the figure in the text by putting figure: *filename* on a line by itself. The *filename* may include the file extension or leave it off.

If multiple figures are included on consecutive lines, the script groups them together in a single row in the web page. For example:

```
figure:foothill larkspur
figure:foothill larkspur spur
```

The above txt results in the following browser formatting:



Formatting Within a Line

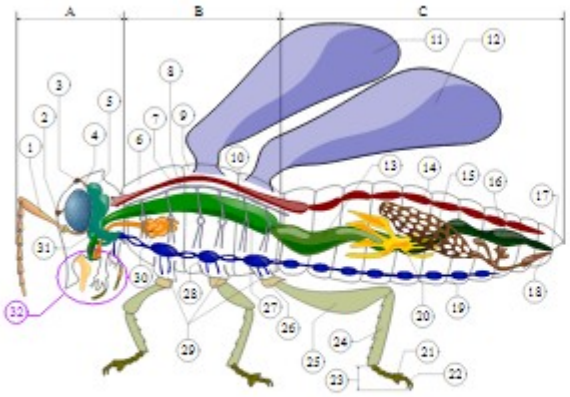
Figure References

A link to a figure can be created by adding [figure:*filename*] to the txt. The *filename* may include the file extension or leave it off. This link can be inline with other text. The BAWG script turns that into [figure] in the HTML with a link to the named figure.

You can use different link text for the figure by using [figure: *filename*,*link text*]. In this case, the link will be created with the text [*link text*]. For example:

```
figure:fly side
A hindwing is either of the rear pair of wings, indicated in the figure as [figure:fly side,part
12].
```

The above txt results in the following browser formatting:



A hindwing is either of the rear pair of wings, indicated in the figure as [part 12].

Although the above example shows both an embedded figure and a figure reference on the same page, they do not have to be used together. Either can be used independently of the other.

Curly Quotes

The script automatically converts an ASCII single or double quotation mark into a corresponding mark curled in the appropriate direction.

- A single quote (') is converted into a right single quotation mark (’) (’ in HTML).
- A double quote (") after whitespace is converted into a left double quotation mark (”) (” in HTML).
- Any other double quote is converted into a right double quotation mark (“) (“ in HTML).

```
An iNaturalist user tells me, "Ishkara's Bloom hasn't been seen in the bay area since the '80s."
```

The above txt results in the following browser formatting:

An iNaturalist user tells me, “Ishkara’s Bloom hasn’t been seen in the bay area since the ’80s.”

The BAWG script is careful to not touch quotes inside an HTML tag.

Common Characters from Jepson

Jepson commonly uses a number of special characters which I’ve found useful to replicate. The BAWG script converts these ASCII representations into the surperior HTML representations.

- +- is converted into ± (± in HTML).
- -- is converted into an en dash (–) (– in HTML).
- <= is converted into ≤ (≤ in HTML).
- >= is converted into ≥ (≥ in HTML).
- << is converted into ≪ (≪ in HTML).
- >> is converted into ≫ (≫ in HTML).

In addition to - above, any single - surrounded by digits (e.g. “2-5”) is also replaed with “–”.

Protecting Angle Brackets

A left angle bracket (<) normally opens an HTML tag, so to get an angle bracket in the text, HTML requires <. To save you pain when you are writing the txt, the script automatically translates a left angle bracket followed by a space (<) into the appropriate HTML (<).

HTML prefers that a right angle bracket (>) is also encoded (as >). However, a right angle bracket that is not within an HTML tag is unambiguously a regular right angle bracket, and all browsers display it correctly. Therefore the script doesn’t do any translation of these.

Links to Other Taxon Pages

A link to another taxon can be created by adding {-*taxon*} to the txt. This link can be inline with other text. The BAWG script turns that into a link with both the common and scientific names listed (if known). For example:

```
See also {-california fetid adderstongue}.
```

The above txt results in the following browser formatting:

> See also california fetid adderstongue (*Scoliopus bigelovii*).

Links to External Sites

I prefer that links to external sites open a new browser tab. But I hate writing all of the HTML cruft necessary to do so safely. Therefore, the script does it for me (and for you).

If <a href="http... appears in the txt, the script inserts target="_blank" rel="noopener noreferrer" into the link tag.

The script assumes that a manual HTML link to another page within the guide uses a relative URL, so the href= portion does not begin with http. In this case, the script doesn't insert any text, so (unless manually directed otherwise) the link opens within the existing browser tab rather than opening a new tab.

Page Name Attributes

"Attributes" declare information about a page. The script recognizes these in the txt, updates the page information, and suppresses the attribute declaration from the HTML output.

Each attribute declaration must appear on a line by itself. The attributes for names are as follows. Additional attributes are described on page 51.

Common Name (com)

The common name for a page is declared as follows:

com: *name*

The name can include spaces. The script theoretically supports common names with uppercase letters, but it has been mostly tested with only lowercase common names.

If the common name is not declared with the com attribute, it can alternatively be inferred from any of the following sources (listed from most to least preferred):

- A child declaration (page 33) by this page's parent that matches the page's scientific name.
- An iNaturalist observation (page 37) that matches the page's scientific name.
- The filename of the page's txt file, but only if it is all lowercase.

Scientific Name (sci)

The scientific name for a page is declared as follows:

sci: *name*

The name can be an elaborated or stripped name, although elaborated is preferred.

If the scientific name is not declared with the sci attribute, it can alternatively be inferred from any of the following sources (listed from most to least preferred). Or if a stripped name is given, the elaborated name can be assigned by one of these:

- A child declaration (page 33) by this page's parent that matches the page's common name.
- An iNaturalist observation (page 37) that matches the page's unique common name.
- The filename of the page's txt file, but only if it is capitalized like a scientific name.

Child Links

A txt file can declare a page as its child by putting ==*name* on a line by itself. If the child does not already exist, it is created.

The specified *name* of the child can be its scientific name (elaborated or stripped) or its common name (if it is unique).

If the child is declared with ==*com*:*sci*, it looks for a page that matches either the specified common name (*com*) or scientific name (*sci*). If a page is found which only specifies one of those names, the other name is filled in from the child declaration. If no page is found, one is created with both names assigned.

A link to the child page is also created. Regardless of how many names were used to create the link, the link displays the common name and scientific name of the child page in the HTML.

Abbreviated Scientific Name

For a child at the species level or below, the scientific name can be abbreviated since the script can figure it out from the parent page. E.g. within the page for "genus Eschscholzia", a child declaration of "==E. californica" is understood to refer to "Eschscholzia californica".

A subspecies or variety can be be abbreviated simply as ssp. *name* or var. *name*.

Key Text

If there is additional text in the same paragraph as the child declaration (on lines following the child declaration, not separated by a blank line), the link and the additional text are formatted as a unit. The additional text is called "key text", and is typically a bullet-point list of distinguishing features for the child compared to other children on the same page. On a wide browser, the key text appears next to the image and below the name; on a narrow browser (such as a phone), the key text appears below the image and name.

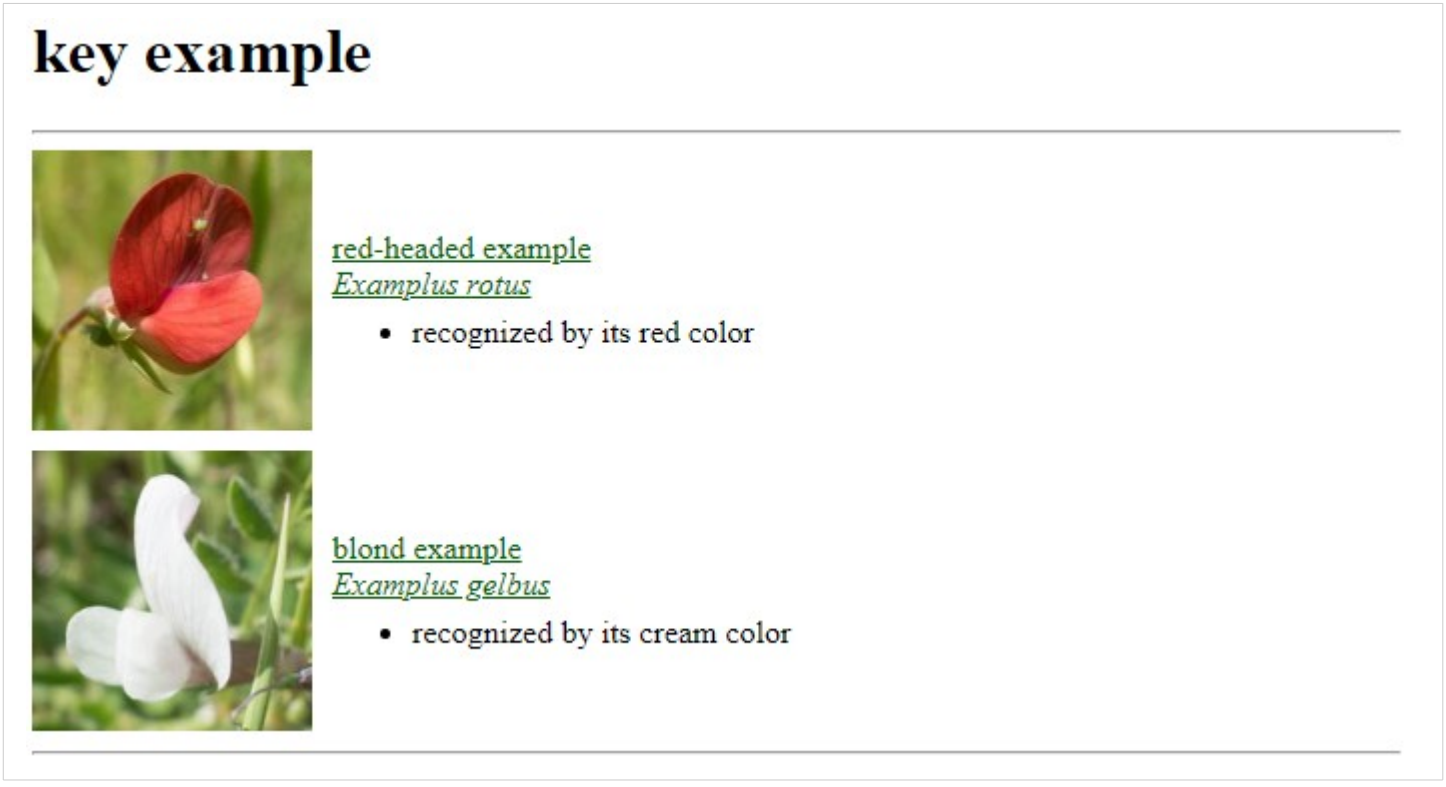
Example with key text:

```
com: key example

==red-headed example:Examplus rotus
. recognized by its red color

==blond example:Examplus gelbus
. recognized by its yellow color
```

The resulting page uses large images to provide plenty of room for the adjacent key text.



Key Page vs. List Page

If key text is specified for any of a page's children, then the page is considered a "key" to aid the user in distinguishing child taxons. However, if a page has children but specifies key text for none of them, then the page is just a "list" of child taxons. A key page is formatted differently than a list page.

In a list page, the order of its children is assumed to not be vital. This often occurs when its child links are created automatically (and are thus in a random order), but it can also occur for manually created child links that have no additional information. Regardless of the cause, if a page includes no key text, consecutive children in its txt are sorted by observation frequency (or alphabetically among children that are observed the same number of times or not at all). If manual child links are interspersed with other text, children are only reordered to the extent that they are bounded by other text.

A list page also uses smaller images to avoid overwhelming the small amount of text for each child.

Example without key text:

```
com: list example

==red-headed example:Examplus rotus

==blond example:Examplus gelbus
```

The resulting page uses smaller images and sorts the child listings.

Child Photo

The child link also displays the first image from the child page (a photo or a CalPhotos placeholder) if it has one. To select a different image to represent the child, the photo suffix (page 24) can be included after either the

common name or the scientific name, e.g. ==*name*,*suffix*. In this case, the photo with the specified suffix is used as the link image instead of the default first image.

Representative Photo

If the page doesn't have any explicitly associated photos, but it has at least one descendent with a photo, the photo used to represent one of the page's children is additionally used to represent the page in any parents' links. By default the page uses the photo from the page's first child that has a photo. But if any child declaration includes an asterisk (*) before the name (e.g. ==**name*), the photo from that child is used to represent the page.

Alternatively, a representative photo is taken from a specified child if the page has a rep attribute (page TBD).

Other Txt Attributes and Properties

Txt can declare other attributes (page 51) and properties (page 45). Before we get to those, however, it is useful to discuss the final source of taxonomy information, iNaturalist observations.

iNaturalist Observations

The iNaturalist web site provides a way to export your observations to a CSV file. The BAWG script can import this file in order to annotate the generated HTML pages with your observations. It can also use extra detail from iNaturalist to get more information about the taxonomy of all observed taxons.

iNaturalist's exprted CSV file should be put in `data/observations.csv`

How to Export iNaturalist Data

When you are logged in to iNaturalist, you can export your observations from this page: https://www.inaturalist.org/observations/export

Fill in the sections of the form as described below.

Section 1: leave the defaults (blank fields and "any" type) except as follows:

- User: your iNaturalist user name

Section 3: clear all checkmarks, then add checkmarks only for the following items. Each of these checkbox names exactly matches a resulting column header in the exported CSV file:

- Basic:
  - id
  - observed_on
  - quality_grade
  - captive_cultivated (although the script doesn't use it yet)
- Geo:
  - place_guess
  - private_place_guess
- Taxon:
  - scientific_name
  - common_name
  - taxon_id
- Taxon Extras:
  - taxon_kingdom_name through taxon_genus_name
  - taxon_species_name
  - taxon_subspecies_name
  - taxon_variety_name
- Observation Fields:
  - none

Finally, click the `Create Export` button at the bottom. This may take a while; if it takes long enough, the web page will direct you to return later to download the CSV file.

How the iNaturalist Data is Used

It is useful to understand how the iNaturalist data can be used to add more information to the database and to reduce the number of txt files that you need to create. Some or all of this data could perhaps be equivalently fetched via iNaturalist's API, but that is not what is currently done.

Note that most of these fields can be blank if there is no corresponding data in the observation or if the field doesn't make sense for the observed taxon.

Keep in mind the distinction between iNaturalist's database and the BAWG database/pages.

Identification and Hierarchy

A page is found or a shadow page is created for each observed taxon, and the observation is counted for that page. The script also ensures that pages and page relationships exist for every level of taxonomy leading to the observed taxon, creating shadow pages and shadow links as necessary.

id

The id field is the numeric ID of the observation in iNaturalist's database. The script doesn't use it for anything, but it can be handy if you notice something odd in the data and want to query iNaturalist for more information about the observation.

scientific_name

The scientific_name is iNaturalist's scientific name for the observed taxon.

The name is stripped of elaborations; i.e. it doesn't include the taxon's rank or distinguish between a subspecies vs. a variety. The script can often distinguish its rank or type by the taxon_*rank*_name fields, below.

The scientific_name is usually the easiest way to match an observation to a page. Although iNaturalist supplies a stripped name without including the rank, the Latin suffix generally makes each name unique even when different ranks are based on the same root word. The exceptions I have noticed occur below the genus level. E.g. "genus *Allium*" includes "subgenus *Allium*", which includes "section *Allium*", which includes "subsection *Allium*". In this case, the taxon_id can be used for a unique match.

In case iNaturalist uses a different scientific name for a taxon than is used by the BAWG, it checks for a match against the alternative scientific name specified with sci_i (page 55).

If a match is found via taxon_id or common_name on a page without a scientific name, the obs_fill_sci property (page TBD) can be used to fill in the page's scientific name from iNaturalist's scientific_name.

taxon_id

The taxon_id field is the numeric ID of the taxon in iNaturalist's database.

The taxon_id is typically used to generate a more precise external link from an HTML page to the corresponding iNaturalist page (page 54).

If the rare cases where iNaturalist scientific_name (above) may be ambiguous, the script looks for a match between iNaturalist's taxon_id and the taxon_id attribute specified by the txt (page 54) to find the correct page.

common_name

The common_name is iNaturalist's scientific name for the observed taxon.

iNaturalist's common name have somewhat random capitalization. They are converted to lowercase to match the scheme for common names used within the BAWG.

The common_name is only used to match an observation to a page if that page has no assigned scientific name.

If a match is found via taxon_id or scientific_name on a page without a common name, the obs_fill_com property (page TBD) can be used to fill in the page's common name from iNaturalist's common_name. Or if the match is made on a page with a different common name, the obs_fill_alt_com property (page TBD) can be used to fill in the alternative name used by iNaturalist.

taxon_subspecies_name and taxon_variety_name

Because iNaturalist never writes an elaborated name to observations.csv, whether a low-level taxon is a subspecies or a variety cannot be immediately determined from its name. However, it writes a name to the taxon_subspecies_name field only if the taxon is a subspecies and to the taxon_variety_name field only if the taxon is a variety. The script checks whether one of these fields is non-empty and elaborates the name accordingly.

taxon_*_name for higher levels

The taxon_*rank*_name fields provide the a (fairly) complete taxonomic hierarchy for the observed taxon, from kingdom down to the observed rank.

The script creates shadow pages and links for each taxon in this hierarchy (if corresponding real pages and/or don't already exist). This is an excellent way to fill in the upper parts of the taxonomy without needing to write a lot of txt files. The shadow pages and links can be promoted to real pages and links using the create and link properties (page 46).

The observed taxon is also repeated in the taxon_*rank*_name, as long as the observed taxon is in one of the listed ranks. If the taxon's rank is already known, the script infers its rank from the lowest-level field that is

filled in, as long as the field shows a matching name. If the script still can't figure out the observed taxon's rank, and it can't find a matching real page, it proceeds on the assumption that the rank isn't important, that you'll never promote the taxon to a real page, and that you only want to include the observation in a higher-level page. Thus, the script creates the page at the lowest rank.

Observation Details

The script gathers additional observation details that may occasionally be useful. By default these are hidden in the HTML, but the user can click "[show details]" to see this additional information:

Chris's observations: 7 (all are research grade) [hide details]

Locations:

- Grant: 2
- Long Ridge
- Monte Bello
- Mt. Umunhum
- Russian Ridge
- Sierra Vista

Months:

- Sep.: 3
- Oct.: 3
- Nov.: 1

quality_grade

The quality_grade field holds one of the following values:

- research: the observed taxon is wild, well observed, and its identification has sufficient agreement.
- needs_id: the observed taxon is wild and well observed, but its identification requires more agreement.
- casual: the observation is not wild or is missing information such as date or location.

The script indicates in the HTML how many of a taxon's observations are research grade versus not.

captive_cultivated

The captive_cultivated field is true for an observation of a captive animal or cultivated organism, false for a wild individual. The script currently doesn't use this field. Since this is the same as a quality_grade of casual (above) for a well-formed observation, perhaps it will never be needed.

place_guess and private_place_guess

Although iNaturalist records GPS coordinates for each observation, we don't need that level of detail for the BAWG. Instead we just want to know the local regions (e.g. parks) where each taxon has been found.

The place_guess field provides the location name for the observation. This location name was either typed in for the observation or inferred by iNaturalist from its GPS coordinates.

If an observation's location is obfuscated (e.g. because it is an endangered species), the place_guess field is also obfuscated. However, if you exported your own observations, the script can get the true location from the private_pace_guess field. Note that private_pace_guess is blank if the location isn't obfuscated.

The script counts the observations within each region and sorts them in the HTML from most to least common.

Region names can be abbreviated or combined using the parks data described on page 43. Observations outside the area of interest (e.g. the SF bay area for the BAWG) can also be discarded if desired.

observed_on

The observed_on field holds the date at which the taxon was observed.

The script writes only the month information to the HTML in order to provide a gauge of seasonality. It tries to identify the "off season" based on consecutive months without observations or months with the fewest observations, and it excludes off-season months with no observations.

Taxon Names Data

When the taxonomy is generated from iNaturalist observations, unless a higher-level taxon happens to have a (poorly identified) observation, iNaturalist supplies only the scientific name. It is often the case that the script can generate all of the other useful information for the taxon's page, but not its common name. Rather than creating a txt file for each of these taxons just to supply the common name, you can simply specify the name mapping in data/taxon_names.yaml

YAML files are in UTF-8 format.

You can look up the YAML standard, but it's pretty trivial to write. A colon (:) associates the thing after the colon with the thing before the colon. These things can be placed on separate lines, with indentation delimiting scope. Strings only need to be enclosed in quotation marks if they use one of these symbols in a potentially confusing manner. Blank lines are ignored.

Taxon names are grouped in the file by rank. Each rank is associated with a number of name pairs, and each name pair associates a (stripped) scientific name with a common name. E.g.

```
family:
  Aizoaceae: stone plant family
  Amaranthaceae: amaranth family
  ...
```

Each name association creates a corresponding shadow page with those names.

Parks Data

The location of each iNaturalist observation is mapped to a park name, and the count of observations within each park is shown in the taxon's observation details.

This section refers to "parks" because those are the regions where I make the most observations. However, you can group your own observations into any regions you choose. Regions can even overlap as long as they are labeled distinctly in the iNaturalist data.

Parks data shoud be put in the file data/parks.yaml

YAML files are in UTF-8 format.

You can look up the YAML standard, but it's pretty trivial to write. A colon (:) associates the thing after the colon with the thing before the colon. These things can be placed on separate lines, with indentation delimiting scope, or they can be on one line if surrounded by curly braces, e.g. {*thing*: *thing2*}. A hyphen (-) at the beginning of a line introduces an element of a list. Strings only need to be enclosed in quotation marks if they use one of these symbols in a potentially confusing manner. Blank lines are ignored.

Parks are grouped into areas, with the "bay area" currently being hard-coded as the area of interest. Each area is followed by a list of parks. E.g.

```
bay area:
- Russian Ridge
- Huddart
- ...

Switzerland:
- Lauterbrunnen, Switzerland
- Törbel, Switzerland

...
```

I should note that the YAML file is organized in a way that is easy for a person to write, not in a way that is directly useful to the script. The script rearranges the data from the file to fit its own purposes.

For each observation, the script checks for a match by any of the parks in the YAML file. E.g. if the observation's location is "Russian Ridge Open Space Preserve, San Mateo County, CA, USA", it matches "Russian Ridge", and so the details for the observed taxon include "Russian Ridge". Since the park name in the YAML file can be any subset of the observed location, the YAML park name effectively acts as an abbreviation the observed location.

If you'd like to use a park name that isn't an exact subset of the observed locations, you can associate a separate search pattern with the park. For example, the following maps a observed location that contains either "Henry Coe" or "Henry W. Coe" to the park abbreviated "Henry Coe".

```
- Henry Coe
- {Henry Coe: Henry W. Coe}
```

An associated search pattern can include any python regular expression syntax. E.g. to search only for an observation that begins with "Henry Coe", use "- {Henry Coe: ^Henry Coe}".

The parks are searched in the order that they are written in the YAML. For example, the following YAML code splits observations in the "Mt. Tamalpais Watershed" from all other "Mt. Tamalpais" locations:

```
- Mt. Tamalpais Watershed
- Mt. Tamalpais
```

Txt Properties

A page can declare properties which apply to that page and/or any choice of taxonomic ranks below that page.

For example, the following can be placed in the top-level page:

```
warn obs_requires_photo: genus-below
```

The above property declaration causes every lower-level page at the genus level or below to flag a warning if the taxon is observed (in the iNaturalist data) but doesn't have an attached photo.

If a property is declared within the paragraph following a child declaration (within the child key), it is treated as being declared by the child. However, a property is typically declared elsewhere in the txt (e.g. at the top) so that it applies to the page itself.

The syntax of a property declaration is *action property_name: ranks*.

Property Name

The property name determines the behavior of the property. Property names are described in the below sections.

Property Action

Depending on the property, the following actions are supported:

- do: Do an action associated with the property when its condition is met. If no action is listed, do is implied.
- caution: Write a cautionary statement into the HTML when the property condition is met.
- warn: Print a warning to standard output when the property condition is met.
- error: Print an error to standard output when the property condition is met. The script also stops if it encounters too many errors (from properties or otherwise).

If a property supports the warn action, it also supports the error action, and vice versa. This document uses the generic *flag* action to mean either warn or error.

Property Ranks

Any number of ranks can be listed. A range can be specified with a hyphen, e.g. "genus-below". The range may be specified as high-low or low-high. Separate ranks or ranges can be separated with commas, e.g. "family, genus-below".

An individual rank can be any of those allowed in a scientific name (page 18), i.e. kingdom through genus.. The rank can also be species or the special keyword below, representing a subspecies or variety. In addition, the special rank self applies the property to the page itself, regardless of its rank. This works even if the page has no

scientific name and thus no rank. The property is never applied to other unranked pages, even if they appear to be within the designated range.

The hyphen for a range can include a slash (/) on either side to exclude the rank on that side from the range. E.g. “self/-below” applies the property to all ranked pages below this one, but not including this page. This can also be helpful if you don’t remember all of the obscure ranks or if more ranks might be added to the list, e.g. “family-/genus” applies from the family level down to but not including the genus level.

Multiple Uses of a Property

In many cases, a property only needs to be listed once with the appropriate ranks in order to be applied everywhere as desired. In other cases, however, it may be desirable to specify a property separately for multiple taxons with a smaller set of ranks for each. This allows the property to be applied more precisely to the desired taxons. In the most extreme case, you may choose to apply a property only to rank self, specifying it exactly on the desired pages.

If a property is specified in one page and then again in a descendent of that page, the higher level property is descarded once it reaches the descendent that re-specifies the property. I.e. the higher-level property applies to the upper taxonomy and any descendents that don’t re-specify the property.

Properties to Create or Link Pages

create

do create promotes each shadow page at the designated ranks to a real page, but only if the shadow page has a real page as a descendent. I.e. this property fills in the hierarchy at the designated ranks above existing pages.

E.g. “do create: family” creates a real page for each taxonomic family that contains a real page (as long as the family is known).

do create also links each created page to its nearest real descendents. E.g. if a taxon is created at the family level, its taxonomy may include shadow pages at the subfamily level, but these are not linked unless something promotes them to be real pages. Instead, the created taxon links to its real descendents, e.g. genus or species pages.

Pages are promoted starting from the bottom of the taxonomy, so when a link to the nearest real descendent is made, the appropriate pages have already been promoted to real.

link

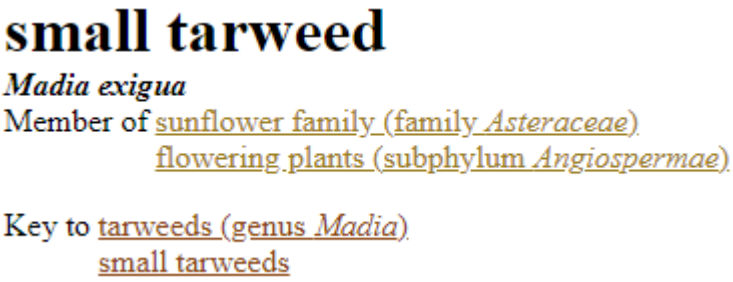
do link links each real page to its nearest real descendents. This is much like create, above, except that it operates on a page that is already real, rather than promoting a shadow page.

member_link

By default, each page links to its immediate real parent(s). If the parent is a key page (page 35), the parent is listed as a key. Otherwise, the page is simply listed as a member of the parent's taxon.

do member_link allows a page to be listed as a member of additional taxons higher in the taxonomy. Specifically, a page lists itself as a member of an ancestor's taxon if the ancestor is a real page and if member_link is applied to the ancestor's rank.

For example, the BAWG includes "do member_link: self, family" in "flowering plants.txt". Thus, each descendent of flowering plants lists itself as a member of flowering plants and of the appropriate family, as below:



Note that "small tarweed" is unusual in that it has two parents, each of which provides key text to help distinguish the taxon from other similar taxons.

member_name

do member_name causes a page to list itself as a member of an ancestor's taxon if the ancestor is a **shadow** page and if member_name is applied to the ancestor's rank.

Unlike member_link, above, the ancestor is only named, not linked, since there is no associated HTML page for the ancestor.

Properties to Check Basic Page Info

no_com

flag no_com flags a real page that doesn't have a common name assigned or disclaimed with n/a (page 20).

no_sci

flag no_sci flags a real page that doesn't have a scientific name assigned or disclaimed with n/a (page 19).

one_child

flag one_child flags a real page that has only one real child. I.e. it flags an extra layer of real hierarchy that isn't useful for distinguishing taxons.

Properties to Link to External Sites

The script can link to external sites at the bottom of the web page for selected taxons, e.g.

Taxon info: iNaturalist – Calflora – CalPhotos – Jepson eFlora

Bay Area species: iNaturalist – Calflora

do link_inaturalist creates a link to the corresponding taxon in iNaturalist from any page to which it is applied. The link uses the taxon_id (page 54) if it is known; otherwise the link triggers a search on the scientific name. If iNaturalist uses a different scientific name, an alternative name can be supplied with sci_i (page 55).

do link_calflora creates a link to the corresponding taxon in Calflora from any page to which it is applied. Calflora supports this type of link at the family level or at the genus level and below. If Calflora uses a different scientific name, an alternative name can be supplied with sci_f (page 55).

do link_calphotos creates a link to the corresponding taxon in CalPhotos from any page to which it is applied. CalPhotos supports this type of link at the genus level and below. If CalPhotos uses a different scientific name, an alternative name can be supplied with sci_p (page 55).

do link_jepson creates a link to the corresponding taxon in the Jepson eFlora from any page to which it is applied. Jepson supports this type of link at the family level or at the genus level and below. If Jepson uses a different scientific name, an alternative name can be supplied with sci_j (page 55).

do link_birds creates a link to the corresponding taxon in AllAboutBirds from any page to which it is applied. AllAboutBirds is indexed by common name, so the scientific name doesn't matter for this application, and only taxons with a common name (i.e. at the species level) can link to AllAboutBirds. There is no way to specify an alternative common name for AllAboutBirds because so far I haven't noticed that one is ever needed.

Note that there is no property for linking to BugGuide. Linking is automatic whenever a bug ID is given.

Link to Bay Area Species

The following properties create a link to find member taxons in the SF Bay Area within the external site's database. When adding a new species to the BAWG, I'll commonly add the species, run the script, then use one of these links to find similar local species that should be added to a key page.

When one of these properties is applied below the genus level, the script automatically promotes the link to the genus level. E.g. the link from a species page will search Calflora for all species in the same genus.

do link_bayarea_inaturalist creates a link from any page to which it is applied to an iNaturalist list of bay area species within the corresponding genus (or higher-level taxon). iNaturalist supports this type of link at any taxonomic level. The link is created by taxon ID or scientific name in the same way as for link_inaturalist, above.

do link_bayarea_calflora creates a link from any page to which it is applied to a Calflora map of SF Bay Area species within the corresponding genus (or higher-level taxon). Calflora supports this type of link at the family level or at the genus level and below. The link is created by scientific name in the same way as for link_calflora, above.

Properties to Count iNaturalist Observations

By default, only a local (bay area) non-casual observation is counted, and only if the observation exactly matches a real page. However, additional observations can be counted by using the following properties.

If an observation's taxon matches a shadow page, the script promotes the taxon up the Linnaean hierarchy to find the first real ancestor page. The properties of that page are then checked to see if the observation is allowed to be counted at the promotedrank. Alternatively, a taxon can be discarded from the count if it appears in ignore_species.yaml (page TBD).

do casual_obs allows a casual observation (page 40) to be counted. Otherwise, only a research or needs_id observation is counted.

do obs_promotion allows an observation to be counted at its promoted rank. See also *flag* obs_promotion, below.

do outside_obs allows an observation outside the bay area to be counted. Whether an observation is within the bay area is determined by matching its observed location to an area in the parks data (page 43). The outside_obs property allows bay-area taxons (e.g. species with real pages) to be counted as observed even if found outside the area. By default this counts only taxons that exactly match a real page. If the observation requires promotion, it is also subject to outside_obs_promotion, below.

do outside_obs_promotion allows an observation outside the bay area to be counted at its promoted rank. This requires outside_obs and obs_promotion to also be applied at the promoted rank.

Properties to Check iNaturalist Observations

The following properties flag an error for certain promoted observations.

The intention is to find pages that ought to be real since I've observed the taxon in the bay area. As such, these properties never apply to observations outside the bay area. They apply to casual observations only if the observation would be allowed by casual_obs. On the other hand, these flag properties apply regardless of whether do obs_promotion (above) allows an observation to be counted.

Warning: Some of these properties are applied based on the rank that the taxon is **promoted to**, while others are based on the rank that the taxon is **promoted from**.

flag obs_promotion flags an observation that gets **promoted to** the taxon with the applied property. This is the strictest check; less strict checks are controlled by the properties below.

flag obs_promotion_above_peers flags an observation is **promoted from** a taxon with the applied property, but only if it the following are both true:

- The observed shadow taxon has no real descendents. The presence of real descendents here implies that the observation isn't identified as well as possible, rather than that it is an unexpected taxon.
- The promoted taxon does have real descendents. The presence of real descendents here indicates that lower-level taxons can be recognized, but the observed taxon is unexpected.

flag obs_promotion_without_x flags an observation that gets **promoted to** the taxon with the applied property if the taxon to which it is promoted isn't explicitly marked incomplete. I.e. the taxon must be marked as hist, rare, hist/rare, more, or uncat (page TBD) to avoid a flag. If the taxon's completeness is unknown or is marked as if it should be complete, a flag is thrown.

Note that obs_promotion_without_x can't catch an error if the observation is promoted to a higher-level taxon than is checked. E.g. if a genus has only one species and so the guide doesn't bother to include a page for the genus itself, an observation of a different species may be promoted above the genus level and so won't flag an error via this property. However, obs_promotion_above_peers can catch this case.

do casual_obs_promotion disables all of the '*flag* obs_*' flags for an observation that isn't research grade. (This includes not only casual observations, but also those that need ID verification.) As with the properties above, this property is applied based on the rank that the observation is promoted to. However, since research grade only makes sense at the species level or below, this property is ignored if the original rank is higher than the species level. The intent is to not force a page to be documented (and thus potentially require a photo & color associated with it) if there isn't enough evidence to be sure which species/subspecies has actually been observed.

Txt Page and Child Attributes

The attributes in this section can apply either to the current page or one of its children. To apply an attribute to a child, include its line in the paragraph following the child link (page 33). To instead apply an attribute to the page itself, you can include it anywhere else in the txt file; the top of the file is typical.

Photo from CalPhotos/Calflora (https://...)

Instead of or in addition to local photographs, the BAWG can also link to photos hosted on CalPhotos or Calflora. In this case, a placeholder is used instead of a regular thumbnail, and the placeholder is linked to the CalPhotos or Calflora photo page.

To include a thumbnail linking to CalPhotos or CalFlora, put the corresponding URL on a line by itself. E.g.

```
https://calphotos.berkeley.edu/cgi/img_query?enlarge=0000+0000+0312+1507
https://www.calflora.org/cgi-bin/noccdetail.cgi?seq_num=gp15975
```

Any number of external photos can be linked this way.

Each photo can also be captioned by preceding its URL with text followed by a colon, i.e. "*text:URL*". E.g.

```
leaf:https://calphotos.berkeley.edu/cgi/img_query?enlarge=4444+4444+0510+1222
red spots:https://calphotos.berkeley.edu/cgi/img_query?enlarge=0000+0000+1107+1653
```

In the above example, the thumbnail placeholders end up looking like this.

Color (color)

One or more colors can be specified for each taxon. The BAWG uses this to describe flower colors.

color: *color list*

The colors in *color list* are separated by commas and can also have whitespace between them.

The list of allowed colors and which part of the taxon hierarchy they can be applied to are determined by the color subset declarations (page TBD).

Ancestor Info

You can specify the name and rank of a higher-level taxon above the current page. I find this to be most useful when I can't (yet) get the hierarchy of a taxon from iNaturalist (e.g. because I haven't observed it or haven't exported my observations yet). There are two methods that can be used to declare an ancestor, described below.

A page can declare any number of ancestors at different ranks. The script infers their proper hierarchy based on their ranks.

Ranked Name (rank)

rank: *name*

E.g.

```
family: Ranunculaceae
```

or

```
family: buttercup family
```

If *name* is a scientific name, and no taxon is yet known, a shadow page is created for it with the elaborated name *rank name*. If *name* is a common name, the taxon must already be known at that rank from some other source (either as a real or shadow page). Declaring an ancestor with no known rank isn't allowed because it would be completely useless with no way to promote it to a real page or otherwise interact with it. The example above using a common name works because the common name can be mapped to a scientific name via taxon_names.yaml (page 42).

Unranked Name (member)

This is much like a child declaration (page 33), but in this case it declares an ancestor. A single common name or scientific name can be specified, or both the common name and scientific name can be specified. The latter is particularly useful for creating a page that otherwise does not exist. In either format, the scientific name can be elaborated with its rank, and its rank must be supplied if the scientific name is ambiguous or the page does not otherwise exist.

member: *name*

member: *com*:*sci*

E.g.

```
member: birthwort family:family Aristolochiaceae
```

Alternative Common Names (acom)

A taxon can have multiple common names. In this case, the script requires one of the names to be the primary common name (set with the com attribute or other method listed on page 32), and the rest are alternative common names (set with the acom attribute).

acom: *name list*

The names in *name list* are separated by commas and can also have whitespace between them. A name man be surrounded with double quotes, which is particularly useful if it includes a comma.

Additional Scientific Names (asci)

(Not to be confused with alternative scientific names, below.)

It is occasionally useful to have a taxon's page include children from other peer taxons. For example, if a genus was recently split, but Jepson still has only a single key in which species from both new genuses are jumbled up, it's easier to put all those species on the same page. Since the page normally has only one genus name, the script will complain that the species with the other genus name can be fit into the taxonomy. You can prevent this error by specifying the scientific name of an additional taxon that the page represents:

asci: *name*

Any number of asci declarations can be given on separate lines. These have no effect on the output of the script; they only tell the script that this page additionally serves as the additional taxon when building the rank-based taxonomy.

Example:

```
com:goosefoots
sci:Chenopodium spp.
asci:Chenopodiastrum spp.

==nettle-leaved goosefoot:Chenopodiastrum murale

==common lamb's-quarters:Chenopodium album
```

```
==pitseed goosefoot:Chenopodium berlandieri

==desert goosefoot:Chenopodium pratericola

==stinking goosefoot:Chenopodium vulvaria
```

Alternative Scientific Names and External Link IDs

(Not to be confused with additional scientific names, above.)

The script can link each taxon page to other websites with useful information about that taxon. Most sites can be searched using a scientific name. However, an external site might not agree with your scientific name for the taxon, or the site might not include the taxon at all. The alternative scientific names specify what name the taxon is used at each external site, if any. These names only need to be specified where they are different from the primary name used for the taxon.

Not all sites agree about the scientific name:

- *Sanguisorba minor* → iNaturalist
- *Sanguisorba minor* ssp. *balearica* → CalFlora
- *Sanguisorba minor* / *Poterium sanguisorba* → CalPhotos
- *Poterium sanguisorba* → Jepson eFlora

The alternative scientific name attributes are sci_i, sci_j, sci_f, sci_p, and sci_b. However, since multiple sites might use the same name, the suffixes can be combined together to specify names for multiple sites at once, e.g. sci_jfp.

The various external sites may use an elaborated name, a stripped name, or a differently elaborated name. For the best results (especially to distinguish ssp. and var.), always specify an elaborated name with these attributes. The script formats the name as necessary for each site. If the alternative scientific name is specified as n/a, the resulting HTML says that it is “not listed” at the corresponding site, but a search link is created in case that ever changes.

Rather than using a search, some sites prefer or require a direct link using a unique numeric ID that the site assigns to each taxon. This ID can be specified for iNaturalist with taxon_id and for BugGuide with bug.

Except for iNaturalist, these names and IDs are only used for external linking in the HTML and not used for any other purpose in the script. E.g. the script can’t look up a taxon using one of these names. The iNaturalist information can also be used for lookups, as explained below.

External Link ID for iNaturalist (taxon_id)

taxon_id: *number*

This specifies the numeric ID for the taxon in the iNaturalist database. This allows the BAWG to link directly to the iNaturalist taxon page, rather than to a page of search results.

The taxon ID can also be retrieved from the iNaturalist data (page 39), but on the rare occasions where the iNaturalist scientific name is ambiguous, setting taxon_id in the txt can help resolve that ambiguity.

Alternative Scientific Name for iNaturalist (sci_i)

sci_i: *name*

This specifies the elaborated scientific name used by iNaturalist. It is used when creating a link to iNaturalist (pages TBD and TBD) or when correlating data from iNaturalist (page 37).

Caution: this works poorly if the iNaturalist scientific name differs from the primary scientific name in more than just the lowest rank. When the script tries to infer the taxonomy from the iNaturalist data, it will conflict with the taxonomy present in the explicit pages.

If the taxon isn't in iNaturalist, the *name* can be set to n/a to prevent an external link from being created. However, this should rarely or never be needed because any taxon with a scientific name should be in iNaturalist somewhere.

Preferably, iNaturalist's taxon ID is known (above) since it allows a link to be made directly to the taxon page. Otherwise, the scientific name is used to trigger an iNaturalist search for the desired page. Even if the taxon ID is known, the name is useful to let the users know where they'll end up.

Alternative Scientific Name for Jepson eFlora (sci_j)

sci_j: *name*

This specifies the elaborated scientific name used by Jepson eFlora. It is only needed when a link to Jepson is created (page TBD). The name can be n/a if the taxon isn't in Jepson (e.g. because Jepson thinks the taxon isn't found in California).

It's a pain to get the numerical ID's that Jepson uses for its pages, so the scientific name is used to trigger a Jepson search for the desired page.

Note that Jepson includes only vascular plants. It can be searched by family, genus, species, subspecies, or variety, but not by any other taxonomic rank.

Alternative Scientific Name for Calflora (sci_f)

sci_f: *name*

This specifies the elaborated scientific name used by Calflora. It is only needed when a link to Calflora is created (pages TBD and TBD). The name can be n/a if the taxon isn't in Calflora (e.g. because Calflora thinks the taxon isn't found in California).

It's a pain to get the numerical ID's that Calflora uses for its pages, so the scientific name is used to trigger a Calflora search for the desired page.

Note that Calflora includes only plants. It appears to include both vascular and non-vascular plants, although the published BAWG currently links to Calflora only for vascular plants. Calflora can be searched by family, genus, species, subspecies, or variety, but not by any other taxonomic rank.

Alternative Scientific Name for CalPhotos (sci_p)

sci_p: *name*

This specifies the elaborated scientific name used by CalPhotos. It is only needed when a link to CalPhotos is created (page TBD). The name can be n/a if the taxon isn't in CalPhotos.

Photos in the CalPhotos database are often tagged with outdated or conflicting scientific names. In order to find as many photos as possible of the desired taxon, you can specify any number of alternative names for CalPhotos by separating them with a vertical bar (|):

sci_p: *name*|*name...*

It's a pain to get the numerical ID's that CalFlora uses for its pages, so the scientific name is used to trigger a CalFlora search for the desired taxons. If alternative names are listed for CalPhotos, the search link searches for the named taxons **and also** the usual scientific name for the page since it may also be useful. However, the HTML only lists the taxons named by sci_f in the browser text.

CalPhotos includes many things, not just plants, and not even just living things. However, the published BAWG currently links to CalPhotos only for vascular plants. CalPhotos can be tagged rather freely, but the convention is generally to label photos of living things with their scientific name. That means that they can be searched by genus, species, subspecies, or variety, but not by any other taxonomic rank. Since a mix of photos of many different species isn't particularly useful, the published BAWG uses CalPhotos only at the species level and below.

External Link ID for BugGuide (bug)

Unlike the other external sites, BugGuide doesn't allow a search query to be triggered by a URL. Therefore, the only effective way to create an external link is with BugGuide's ID number for the taxon.

bug: *number*

This specifies the numeric ID for the taxon in the BugGuide database. This allows the BAWG to create a link to the BugGuide taxon info page.

Alternative Scientific Name for BugGuide (sci_b)

sci_b: *name*

This specifies the elaborated scientific name used by BugGuide. It is only useful when a link to BugGuide is created (page TBD) and BugGuide's numeric ID is known.

Since the link uses BugGuide's numeric ID, the provided sci_b name is only used for display in the browser. And since the link is only displayed if the numerical BugGuide ID is known, there's no need or support for sci_b to be n/a.