Bay Area Wildflower Guide

Revision 1.0 (2021-07-08)

Copyright 2021 Chris Nelson

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 Linnaean 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 Running the Script on page 10.

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/group_names.yaml    # Mapping btwn common and scientific 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
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 children are usually unnecessary.

Photos are normally associated with a txt file or with a child named in the txt. However, if a photo’s filename doesn’t match any known taxon, it creates its own page for the photo. In my current workflow, however, 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 Linnaean 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.

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.)

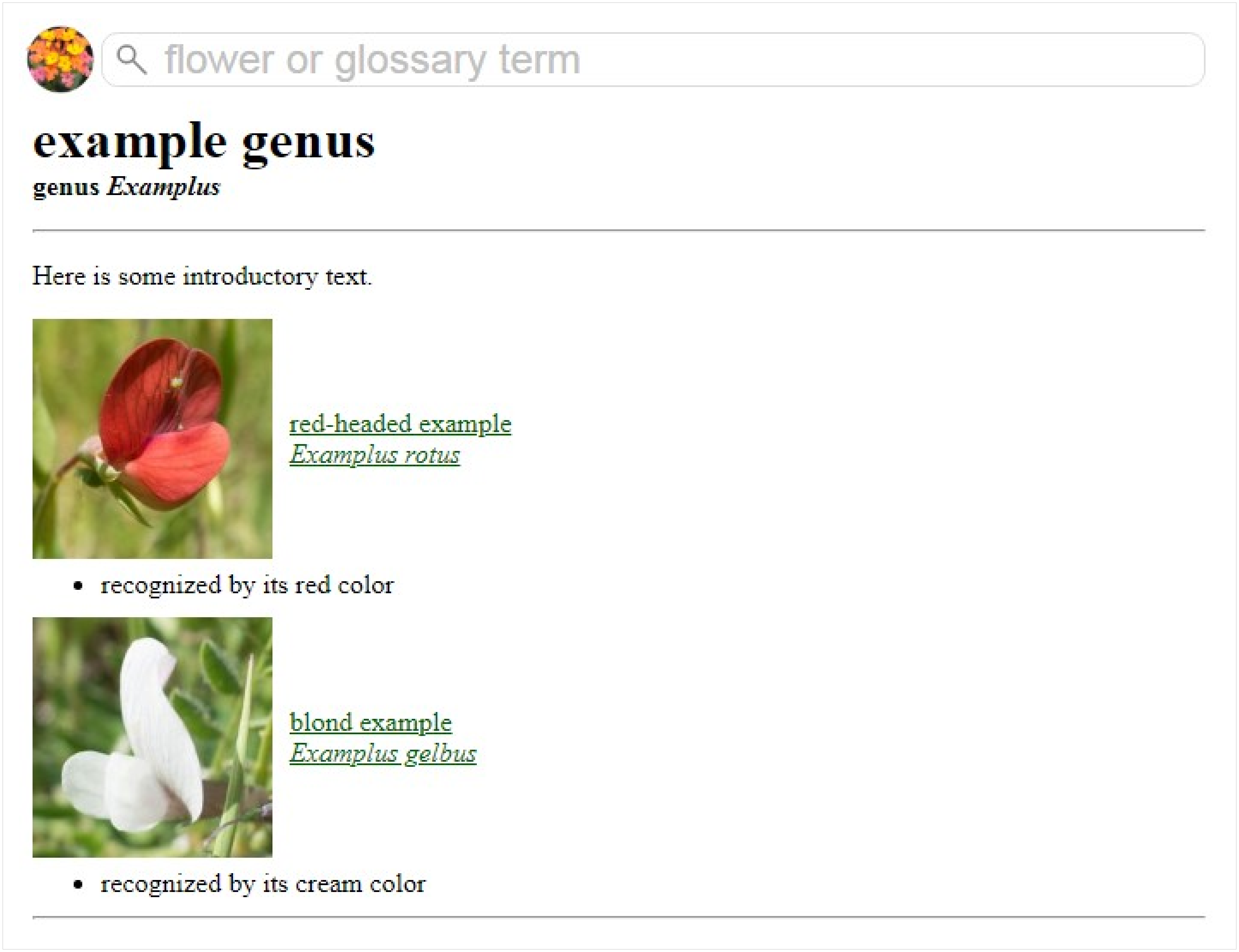
The txt continues with declarations for the two species under the example genus. Each child is declared with both a common name, e.g. “red-headed example” and a scientific name, e.g. “Examplus rotus”. Each child is followed by information that is useful when distinguishing among species within a genus.

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



example genus

genus *Examplus*

Here is some introductory text.

red-headed example
Examplus rotus

- recognized by its red color

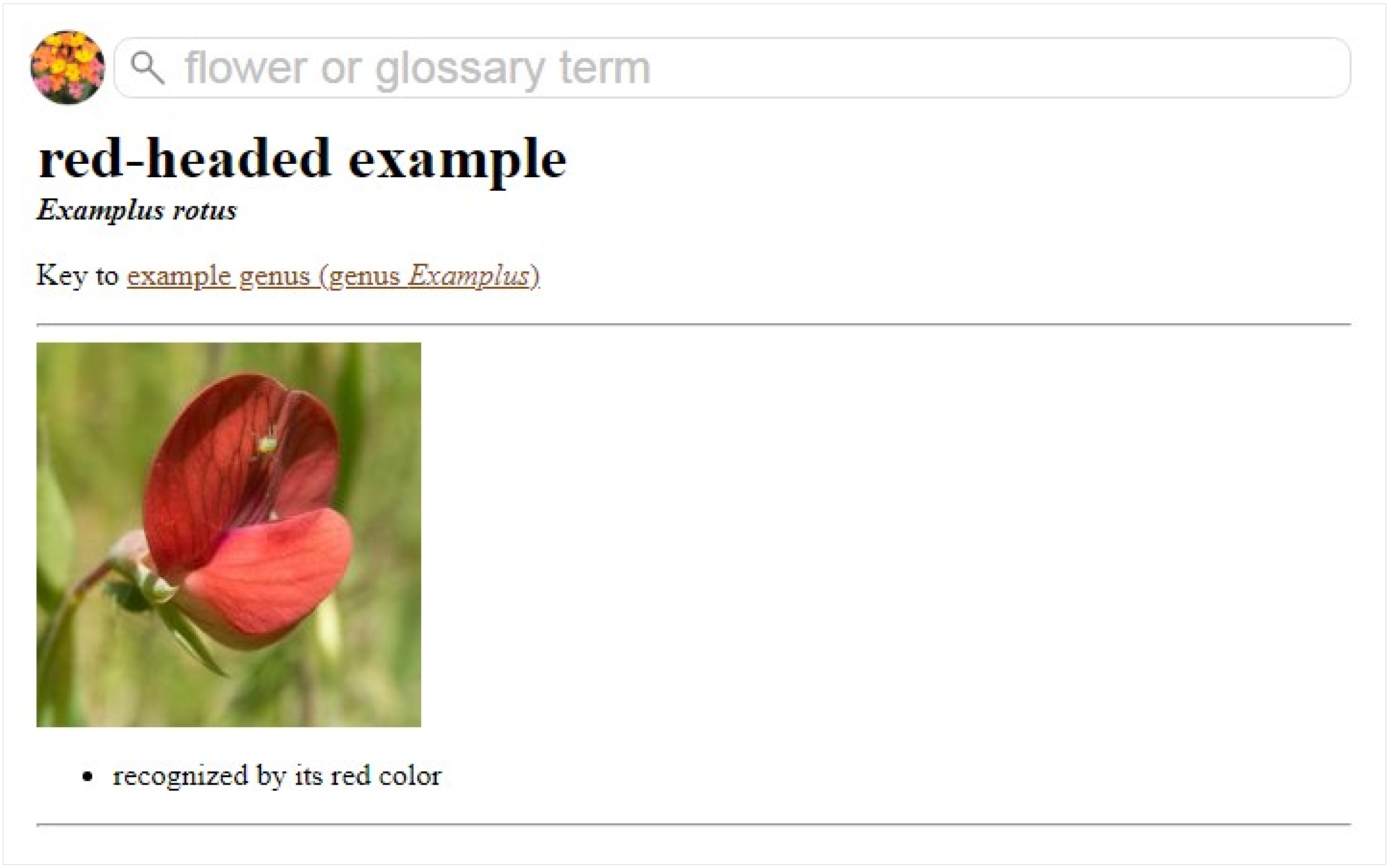
blond example
Examplus gelbus

- recognized by its cream color

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



flower or glossary term

red-headed example

Examplus rotus

Key to example genus (genus *Examplus*)

- recognized by its red color

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.

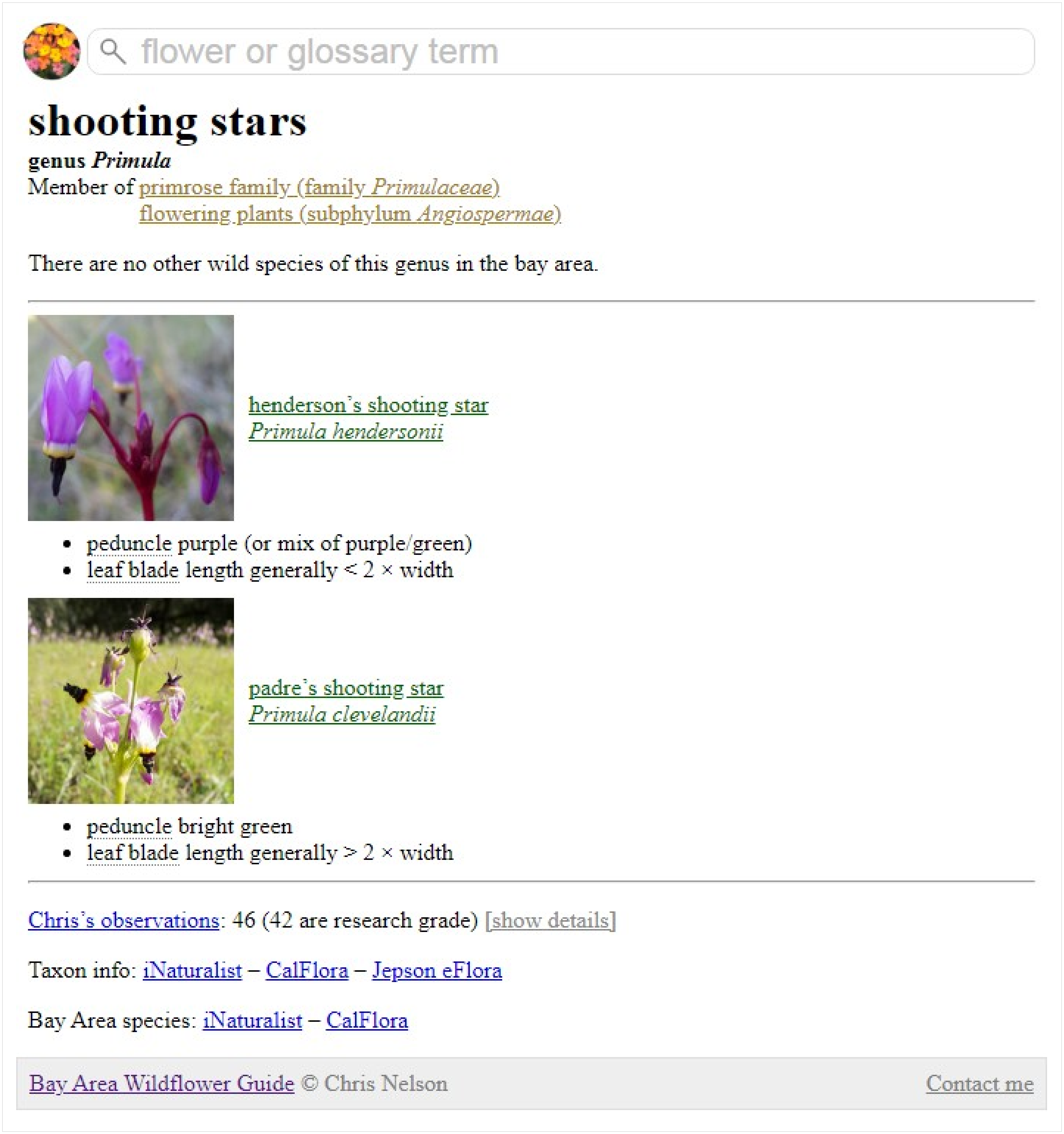
On these child pages, rather than linking the photo thumbnail to another page, the thumbnail links to the original full-size version of the photo.

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



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 the 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.

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 <name>

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 group_names.yaml
Reading data/group_names.yaml
Step 5: Update Linnaean 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: 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 <n>

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, its name is enclosed in [brackets]. Ithe child's name is also prefixed with a symbol indicating the type of parent-child relationship: a Linnaean child only (`-`), a real child only (`+`), or both a real and a Linnaean 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.

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 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.

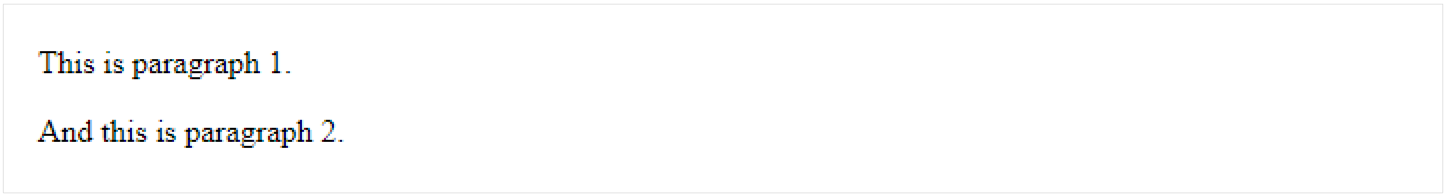
Paragraphs

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 <p> and </p> 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.