**Overview of the the BIEN2 Analytical Database**

Prepared by: Brad Boyle

Date: 28 Aug. 2012

Last revision: 11 Nov 2012

Note: this document replace the previous document " BIEN2 Validation Workflow"

**Contents**

1. Introduction 1

2. Create & populate table `geoscrub` 2

3. Prepare political division authority tables 3

4. Geographic name resolution 4

5. Index New World observations 5

6. Geovalidation 6

8. Detection and flagging of cultivated specimens 7

9. Detection and flagging of FIA plots from plantations and logged areas 8

10. Transfer geovalidation results to bien2 database 8

11. Taxonomic resolution 8

12. Index higher taxa 10

13. Index threatened and endangered species 15

14. Aggregate species abundances from plot observations 15

15. Standardize plot metadata regarding plot methodology 15

16. Index data providers 16

17. Determine a "concensusHabit" (growth form) for each species 16

## 1. Introduction

This document provides an overview of the validations performed during building of the BIEN2 analytical database. These validations are executed primarily in a separate database, `geoscrub`, prior to building the main analytical database table, `bien2`.`viewFullOccurrence`, and the derived analytical database tables in database `bein\_web`. This document does not discuss taxonomic resolution. Taxonomic resolution is discuss briefly in the separate document which provides an overview of the entire BIEN2 analytical database.

The most important validation, geovalidation, involves marking the geocoordinates (latitude, longitiude) assoicated with a particular observation record as invalid (isGeoValid=0) if the point falls outside the boundaries of one or more of the declared political divisions associated with the record (country, state, county). Points falling inside all declared political divisions are treated as valid (isGeoValid=1). Observation with invalid coordinates are excluded from most analyses. Geovalidation does not include georeferencing, which is the determination of new coordinates based on political divisions and locality descriptions. No georeferencing was performed for the BIEN2 database.

Geovalidation is performed using a point-in-polygon approach, in which coordinates are classified as falling in or outside the relevant polygon. For points falling outside the polygon, distance to the nearest boundary is also calculated as a measure of error.

Prior to geovalidation, we first perform what we call "Geographic Name Resolution" (GNR). GNR involves matching the verbatim names of each political division associated with the observation record against standard lists of names and codes for countries, states and counties. We use a large library of standard names and codes, plus alternative names in English, Spanish, French and Portuguese to perform conversion to a single standard name. The GNR scripts also convert utf8 codes, extended ascii codes and extended ascii characters to plain ascii text. Although the GNR scripts do not perform fuzzy matching, they typical result in standardization of the majority of unmatched political division names, thereby enabling geovalidation to proceed for these records. GNR was also performed for the names of political division polygons used for point-in-polygon geovalidation.

After completion of GNR, resolved country names are used to flag whether observations are from the New World (western hemisphere) or Old World (eastern hemisphere). This is done by joining resolved country names to a comprehensive list of New World countries. If the record joins, isNewWorld=1, else isNewWorld=0.

In addition to GNR, geovalidation and flagging New World countries, we also detect and index observations representing cultivated specimens or disturbed plot sampling locations. Cultivated observation exclusion (COE) involves setting boolean field isCultivated=1 (default isCultivated=0, and, if isCultivated=1, populating column isCultivatedReason with the rule used to detect cultivated status. COE validation is necessary because observations of cultivated plants or disturbed communities are normally excluded from our analyses. As COE is performed as part of the geovalidation pipeline, prior to transferring the validated results back to the `bien2` database, it is described here.

All validations described below are performed in database `geoscrub` on nimoy, unless otherwise noted.

Note: this document replace the previous version " BIEN2 Validation Workflow"

## 2. Create & populate table `geoscrub`

* Scripts in: geoscrub/create\_geoscrub/
* Master script: create\_geoscrub.php
	+ calls all others
* Creates table `geoscrub` & populates with verbatim locality information from two locations in bien2 core database:
	+ bien2.IndividualObservation JOIN bien2.SpecimenObservation (specimen localities)
	+ bien2.PlotMetaDataDimension (plot localities)
* Indentifies records by original PK + table name
* Table includes additional fields to hold results of standardization of political division names and validation of geocoordinates
* Transfers original identifiers (table + table PK), verbatim political division names, coordinates, elevations and locality descriptions. Currently, only political division names are standardized, and coordinates validated against the standardized political division names; verbatim elevation and locality\_description fields are not currently used, but were included for forward compatibility for georeferencing (not yet implemented)
* See Fig. 1

## 3. Prepare political division authority tables

* See Fig. 2 for overview of resulting schema
1. **Country**
* Scripts in: geoscrub/create\_country/
* Master script: create\_country.php
* See Fig. 2
* Creates the following tables:
	+ country
	+ countryName
	+ countryStaging
	+ countryNameStaging
* Populates with reference data in files country.txt, countryName.txt
* Includes 2-char and 3-char ISO county codes (stable) in addition to generating artificial auto\_increment PK.
* Staging tables are deleted after normalization
1. **StateProvince**
* Scripts in: geoscrub/create\_stateProvince/
* Master script: create\_stateProvince.php
* See Fig. 2
* Creates the following tables:
	+ stateProvince
	+ stateProvinceName
	+ stateProvinceStaging
	+ stateProvinceNameStaging
* Populates with reference data in files stateProvince.txt, stateProvinceName.txt.
* The name used as the "standard name" is converted to plain ascii (although I notice this is inconsistent for some reason…perhaps the function did not succeed in converting all extended ascii characters?)
* In addition to generating an artificial (auto\_increment) PK (stateProvinceID), includes 2-char stateProvince code (not universal, some missing), HASC\_1 codes (unique but some missing), a "standard" full name (chosen arbitrarily by me) and a unique code consisting of the country ISO code and the stateProvince name joined by an underscore.
* Provinces are joined to parent country initially by ISO codes, and later by the artificially generated countryID
* Determines and adds plain ascii equivalents of political division names
* Staging tables are deleted after normalization
1. **CountyParish**
* Scripts in: geoscrub/create\_ countyParish/
* Master script: create\_ countyParish.php
* See Fig. 2
* Creates the following tables:
	+ countyParish
	+ countyParishName
	+ countyParishStaging
	+ countyParishNameStaging
* Reference data in files countyParish.txt, countyParishName.txt are loaded first to the staging tables, then normalized to the primary reference tables (countyParish, countryParishName)
* The name used as the "standard name" is converted to plain ascii.
* Table `countyParish` consists of an artificial (auto\_increment) PK (countyParishID), a FK linking to the artificial PK (stateProvinceID) of the parent stateProvince, a standard plain ascii version of the name, to be used as the "standard" name, a unique code (made by me) consisting of the country ISO code and the countyParish name joined by an underscore, and the (mostly) stable HASC\_2 code.
* Table `countyParishName` contains synonyms of countyParish names, and consists of an artificial auto\_increment PK (countyParishNameID), the FK linking to the master entry for that political division in table `countyParish` (countyParishID), and countyParishName.
* Staging tables are deleted after normalization

## 4. Geographic name resolution

* Correct, standardize and index names of political divisions country, stateProvince, and countyParish.
* Used for (1) table geoscrub, (2) political division polygons (i.e., names of shapefiles used for point-in-polygon validation), (3) newWorldCountries authority table (used for indexing New World countries; see step 5, below), and (4) indexing countries in table herbaria (used for detecting cultivated specimens; see step 8c, below)
1. **Country**
* Scripts in: geoscrub/scrub\_country/
* Master script: country\_scrubbed.php
* Creates table `countryScrubbed` (see Fig. 2)
* Inserts into above table all unique verbatim country names from table `geoscrub`
* Attempts to link verbatim name to standard country in table `country` by joining by (i) 2-char ISO code and (ii) 3-char ISO code.
* Attempts to link verbatim name to standard name via synonyms, after converting any unconverted utf8, extended ascii and plain ascii codes in name to to plain ascii text
* Transfers results of scrubbing (countryID) to table `geoscrub`
1. **StateProvince**
* Scripts in: geoscrub/scrub\_stateProvince/
* Master script: poldiv\_scrubbed.php
* Creates table ` stateProvinceScrubbed` (see Fig. 2)
* Inserts into above table all unique verbatim country names from table `geoscrub`
* Attempts to link verbatim name to standard stateProvince in table `stateProvince` by joining by (i) independent code (HASC\_1) and (ii) dependent code (must be paired with country identifier).
* Attempts to link verbatim name to standard name via synonyms, after converting any unconverted utf8, extended ascii and plain ascii codes in name to to plain ascii text
* Transfers results of scrubbing (stateProvinceID) to table `geoscrub`
1. **CountyParish**
* Scripts in: geoscrub/scrub\_countyParish/
* Master script: poldiv\_scrubbed.php
* Creates table ` countyParishScrubbed ` (see Fig. 2)
* Inserts into above table all unique verbatim country names from table `geoscrub`
* Attempts to link verbatim name to standard countyParishin table `countyParish` by joining by (i) independent code (HASC\_2) and (ii) dependent code (must be paired with country identifier).
* Attempts to link verbatim name to standard name via synonyms, after converting any unconverted utf8, extended ascii and plain ascii codes in name to to plain ascii text
* Transfers results of scrubbing (countyParishID) to table `geoscrub`

## 5. Index New World observations

* Requires table geoscrub.newWorldCountries (see Fig. XXX). This table was compiled by Rick Condit (not sure of source). Names were manually standardized to match standard country names in geoscrub.country.
* Column geoscrub.isNewWorld updated by a simple inner join on countryID:

UPDATE geoscrub g JOIN newWorldCountries c

ON g.countryID=c.countryID

SET g.isNewWorld=1

WHERE c.isNewWorld=1;

## 6. Geovalidation

* Record identifiers (artificial PK geoscrubID), verbatim coordinates and standardized 1st, 2nd and 3rd political divisions (country, stateProvince, countyParish) are extracted from table `geoscrub`.
* Standardized political division names determined in step 3 above are used to validate or reject coordinates by performing point-in-polygon fits to political division polygons.
* Note that geovalidation **also** requires that names of political division shape files be standardized and linked to the political division authority tables populated in Step 2 above.
* Each point is coded as in or out of its declared political division, and distance from political division polygon in miles is estimated
* See separate document by John Donoghue, "BIEN2 GeoValidation Workflow", for details of point-in-polygon validation procedure
* Results of geovalidation are transferred back by joining results to table to `geoscrub` by geoscrubID.
* Results are stored in table `geoscrub` as the following fields:
	+ **isInCountry**: 0=out, 1=in, NULL=not validated
	+ **distErrCountry**: distance in km from country boundary. Equals 0 if isInCountry=1. NULL if not validated.
	+ **isInStateProvince**: 0=out, 1=in, NULL=not validated
	+ **distErrStateProvince**: distance in km from stateProvince boundary. Equals 0 if isInStateProvince =1. NULL if not validate.
	+ **isInCountyParish**: 0=out, 1=in, NULL=not validated.
	+ **distErrCountyParish**: distance in km from countyParish boundary. Equals 0 if isInCountyParish =1. NULL if not validated.
* NULL values for isInCountry, isInStateProvince and isInCountyParish occur when geovalidation status is unknown because a standard political division could not be assigned to that record.
* As of BIEN2, geovalidation was performed for all countries, and for stateProvince only for the United States, Mexico and Brazil. Geovalidation using countyParish was not performed
* Thus, selecting strictly "geovalid" records from the geoscrub table would require the following WHERE criteria:

WHERE isInCountry=0 AND (isInStateProvince=1 OR isInStateProvince IS NULL)

* Using a more relaxed error buffer of 10 km, the above condition becomes:

WHERE (isInCountry=0 OR distErrCountry<=10) AND (isInStateProvince=1 OR distErrStateProvince<=10 OR isInStateProvince IS NULL)

* An equivalent condition to the preceding for table bien2.viewFullOccurrence would be:

WHERE (countryError<=10) AND (ProvinceError<=10 OR ProvinceError IS NULL)

* And for table bien\_web.observation:

WHERE (isGeovalid=1)

Note that `bien\_web`.`observation`.`isGeovalid` incorporates the complex conditions of the preceding conditions.

## 8. Detection and flagging of cultivated specimens

* Specimens of cultivated plants can grossly distort range models
* These scripts attempt to detect and exclude cultivated specimen observations, and also flag plot observations from plots in human-altered forests such as plantations and logging treatments
* Scripts in: geoscrub/cultivated/
* Results tranferred to bien2.viewFullOccurrence are in geoscrub/cultivated/update\_view/ (not sure about geoscrub!)
* Updates the following two fields in `geoscrub` (transferred to both bien2.viewFullOccurrence and bien\_web.observation):
	+ **isCultivated**: 1=yes, 2=no, NULL=not checked
	+ **isCultivatedReason**: which of four methods below used as basis for setting isCultivated=1
* Four methods:
1. Original "isCultivated" flag. Some databases (e.g., ARIZ herbarium) flag cultivated specimens themselves. This information was not captured in the bien2 database, but could be retrieved in some cases from original data.
2. Keywords in locality and specimen descriptions (see geoscrub/cultivated/cult\_by\_locality/). Wildcard searches in English, Spanish, Portuguese and French for key words equivalent to "cultivated", also "plantation", "garden", "farm", etc.
3. Proximity to herbarium or botanical garden (see geoscrub/cultivated/cult\_by\_herbaria/). Specimens within 3 km radius of a herbarium are excluded. Coordinates of herbaria obtained from Index Herbariorum.
4. Proximity to city (see geoscrub/cultivated/cult\_by\_city/). NOT USED. We had originally intended to exclude specimens within a certain radius of cities (scaled by population) but decided not to use this method due to the high rate of false positives.

## 9. Detection and flagging of FIA plots from plantations and logged areas

* These scripts attempt to detect and exclude FIA plots human-altered forests such as plantations and logging treatments
* Based on fields in FIA\_COND table. FIA\_COND contains metadata on plot conditions, and was obtained from FIA website (http://apps.fs.fed.us/fiadb-downloads/datamart.html)
* Records in table geoscrub are joined to FIA\_COND via original plots codes in table bien2.PlotMetaDataDimension (see Fig. 3).
* Scripts that build table `geoscrub`.`FIA\_COND` are in: geoscrub/fia/
* Updates the following two fields in `geoscrub` (transferred to both bien2.viewFullOccurrence and bien\_web.observation):
	+ **isCultivated**: 1=yes, 2=no
	+ **isCultivatedReason**: if isCultivated=1, this field is set to 'FIA disturbed'

## 10. Transfer geovalidation results to bien2 database

* Validation results are NOT transferred to core bien2 database tables, nor are they transferred directly to analytical database tables bien2.viewFullOccurrence and bien\_web.observation
* Instead, they are transferred to two new tables in bien2:
	+ geoIndividualObservation
	+ geoPlotMetaDataDimension
* These tables are essentially copies of the table geoscrub, with verbatim and validation columns separated out into two separate tables corresponding to the original source tables (see Fig. 4).
* Validation results are incorporated into analytical by Rick Condit's SQL procedure that creates bien2.viewFullOccurrence.
* Scripts:
	+ geoscrub/create\_geovalidation\_tables/
		- Master script: create\_geovalidation\_tables.php
	+ geoscrub/update\_geovalidation\_tables/
		- Master script: update\_geovalidation\_tables.php

## 11. Taxonomic resolution

* Correcting taxonomic names for the BIEN2 analytical database was a huge undertaking which ultimately resulted in construction of the TNRS, or Taxonomic Name Resolution Service (http://tnrs.iplantcollaborative.org/).
* Correction of taxonomic names in BIEN2 begins by extracting all species names from the table TaxonDimension. TaxonDimension is itself a SELECT DISTINCT on all taxonomic names within the core BIEN2 database.
* For every record in TaxonDimension, the contents of the fields FamilyName, GenusName, SpecificEpithet, InfraspecificRank, InfraspecificEpithet and Authority are concatenated with spaces and stored in the column taxonAuthorityVerbatim. The contents of the latter column are then uploaded manually to the TNRS web interface, using the file upload utility. Note that family is pre-prended to the scientific name; this is done to enable use of the "taxonomic constraint" option of the TNRS, which prevents names from "hoppping" between unrelated homonyms. During name resolution, Tropicos is used as the only taxonomic source, partial matching is enable and match sensitivity is left at its lowest, default setting. Once resolution is complete, the Best Match Settings "Constrain by higher taxonomy" is turned on prior to retrieval of the final results.
* The resolved names are manually updated to the table TaxonDimension. Note the following mappings from the TNRS results file to columns of TaxonDimension:

Name\_matched🡪taxonMatched

Name\_matched\_author 🡪taxonMatchedAuthority

Overall\_score🡪matchScore

Taxonomic\_status🡪taxonMatchedAcceptance

Unmatched\_terms 🡪unmatched terms

Accepted\_name\_family 🡪familyCorrected

IF(Accepted\_name\_species IS NULL,NULL,strSplit(Accepted\_name\_species ,' ',1))🡪genusCorrected

Accepted\_name\_species🡪speciesCorrected

Accepted\_name 🡪taxonCorrected

Accepted\_name\_author 🡪authorityCorrected

Accepted\_name\_rank 🡪taxonCorrectedRank

IF(Accepted\_name<>Name\_matched,"Accepted", Taxonomic\_status )🡪taxonCorrectedAcceptance

Unmatched\_terms 🡪morphospecies

IF(Accepted\_name IS NULL, CONCAT(IFNULL(,''),IFNULL(Unmatched\_terms)),

CONCAT(IFNULL(Accepted\_name\_family,''),IFNULL(Accepted\_name),

IFNULL(Unmatched\_terms)) 🡪taxonMorphospecies

* During population of viewFullOccurrence, only the accepted name and name components are used, as shown by the following mappings from TaxonDimension to the columns of viewFullOccurrence:

familyCorrected🡪Family

TaxonomyID🡪TaxonomyID

GenusName🡪OrigGenus

IF(SpecificEpithet IS NULL,NULL,CONCAT(OrigGenus,' ',SpecificEpithet🡪OrigSpecies

taxonCorrected🡪TaxonCorrected

genusCorrected🡪Genus

NULL🡪Species [not used]

speciesCorrected🡪Latin

taxonCorrectedRank🡪Rank

taxonCorrectedAcceptance🡪Accepted

taxonMorphospecies🡪TaxonMorphoSpecies

The preceding mapping is my "ideal" version; this mapping is not followed precisely as shown by the procedures that build viewFullOccurrence. For this reason, during construction of the second generation analytical database, bien\_web, only the foreign key TaxonomyID is transferred from bien2.viewfullOccurrence to bien\_web.observation. The remaining taxonomic columns of table observation are populated in a second step by joining bien\_web.observation.bien2\_taxonomyID ON bien2.TaxonDimension. TaxonomyID as follows:

bien2.TaxonDimension.taxonAuthorityVerbatim 🡪taxonAuthorityVerbatim

bien2.TaxonDimension.familyCorrected 🡪family

bien2.TaxonDimension.genusCorrected 🡪genus

bien2.TaxonDimension.speciesCorrected 🡪species

bien2.TaxonDimension.taxonCorrected 🡪taxon

bien2.TaxonDimension.authorityCorrected 🡪taxonAuthor

bien2.TaxonDimension.taxonMorphospecies 🡪taxonMorphospecies

bien2.TaxonDimension.taxonCorrectedRank 🡪rank

bien2.TaxonDimension.taxonCorrectedAcceptance 🡪acceptance

## 12. Index higher taxa

Many bien analyses require selection of species belonging to particular groups (such as ferns or flowering plants). Also, many observations in the database are of non-plant taxa such as protozoans, fungi or animals, or are of lower plant taxa which we desire to exclude (for example, red algae). Such observations must be excluded from all BIEN analyses. For this reason, we join all species in BIEN to a phylogenetic backbone drawn from the NCBI Taxonomic database (<http://www.ncbi.nlm.nih.gov/taxonomy>) and use this join to populate a column, bien\_web.observation.higherPlantGroup, which identifies the major plant clade to which the species belongs. Once the taxonomic backbone has been prepared, taxa are joined to the backbone by genus, or by family if the observation is identified to family only.

This operation is performed in four steps, as follows.

1. Import NCBI phylogenetic backbone
	1. Files 'names' and 'nodes' are extracted from ' taxdump.tar.gz ' (as imported from the NCBI Taxonomy ftp site, ftp://ftp.ncbi.nih.gov/pub/taxonomy) and imported into MySQL as tables 'cladeName' and 'clade', respectively (Fig. 5).
	2. The mappings to table cladeName are as follows

ncbi.name.tax\_id 🡪 bien2.cladeName. ncbi.name.cladeID ncbi.name.name\_txt 🡪 bien2.cladeName.cladeName

ncbi.name.unique\_name 🡪 bien2.cladeName.cladeNameUnique

ncbi.name.name\_class 🡪 bien2.cladeName. nameStatus

* 1. The mappings from names to table clade are as follows

ncbi.clade.tax\_id 🡪 bien2.clade.cladeID

ncbi.clade.parent\_tax\_id --> bien2.clade.parentCladeID

ncbi.clade.rank --> bien2.clade.rank

The remaining columns in ncbi.names were not used.

1. Add and populate new columns
	1. The following new columns were added to cladeName:

cladeNameID: artificial primary key; integer autoincrement

source: identifies source of names. Values: "ncbi", "tropicos", "grin"

* 1. The following new columns were added to clade, as follows:

cladeNameStd: Copied over from cladeName. Equals cladeName.cladeName WHERE cladeName.nameStatus='scientific name'. There is only one such value for each value of cladeID in table cladeName. Added as a convenience to allow me to query without joining to table cladeName.

source: Copied over from cladeName. Equals cladeName.source WHERE cladeName.nameStatus='scientific name'. Added as a convenience to allow me to query without joining to table cladeName.

isHybrid: Equals 1 if name is a hybrid, else 0. Populated by searching for hybrid "x " or " x " in cladeNameStd. Used as shortcut for excluding hybrids from certain queries.

leftIndex: Modified pre-order tree traversal left index, for ancestor-descendent lookups. See below for details.

rightIndex: Modified pre-order tree traversal right index, for ancestor-descendent lookups. See below for details.

1. Populate modified pre-order tree traversal indices in table `clade`
* Needed to retrieve ancestors or descendant to arbitrary depth in single query
* Requires complete parent-child adjaceny links (true for ncbi table nodes, =cladeName)
* Populated using the recursive php function get\_index() (see Appendix 1).
* Appendix 2 shows example of how this function is called. Includes adding root to table, and correcting for multiple taxa linked directly to root
* See appendix 3 for examples of ancestor and descendant retrieval using these indices.
1. Add and populate higher taxa columns to table `clade`
* The following columns provide a quick way of looking the main higher taxonomic groups to which a particular taxon belongs. They are populated by ancestor lookup using columns leftIndex and rightIndex, and serve as shortcuts to actually doing the ancestor lookup. I use English equivalent names of latin higher clade names (for example, "vascular plants" instead of "Tracheophyta").

`family`: populated with name of family if clade is of rank family or lower.

`majorPlantGroup`: Identifies taxa subtended by node "Viridiplantae" (rank="kingdom"; green plants). Values: "Viridiplantae", NULL.

`embryophytes`: identifies major clades subtended by unranked node "Embryophyta" (land plants). Values: "hornworts", "liverworts", "mosses", "unclassified embryophyte", "vascular plants", NULL. Equals null if taxon is not an embryophyte.

`vascularPlants`: Identifies major clades subtended by unranked node "Tracheophyta" (vascular plants). Values: "ferns and horsetails", "lycophyte", "seed plants", "seed plants and ferns", NULL. Equals null if taxon is not a trachophyte.

`seedPlants`: Identifies major clades subtended by unranked node "Spermatophyta" (seed plants). Values: "conifers", "cycads", "flowering plants", "ginkgos", "gnetophytes", NULL. Equals null if taxon is not a spermatophyte.

`floweringPlants`: Identifies major clades subtended by unranked node "Magnoliophyta" (flowering plants). Values: "basal magnoliophytes", "ceratophyllales", "eudicots", "magnoliids", "monocots", "unclassified magnoliophytes", NULL. Equals null if taxon is not a flowering plant.

`higherPlantGroup`: This column is populated based on the preceding five columns. It classifies all land plants (embryophtes) into higher taxonomic groups most likely to be of interest to users of BIEN. Values: "bryophytes", "ferns and allies", "flowering plants", "gymnosperms (conifers)", "gymnosperms (non-conifer)". Equals NULL if taxon is not a land plant (embryophyte). This is the column which is copied over into the second generation analytical database as bien\_web.observation.higherPlantGroup.

Below are the English-equivalent names used in the last five columns of table clade to the latin higher clade names in column cladeNameStd, and their ranks, if any. You must use the latin names when doing an ancestor or descendant lookup (see Appendix 3).

"green plants": "Viridiplantae", no rank

"land plants": "Embryophyta", no rank

"hornworts": "Anthocerotophyta", no rank

"liverworts": "Marchantiophyta", no rank

"mosses: "Bryophyta", no rank

"unclassified embryophyte": "unclassified Embryophyta", no rank

"vascular plants": "Tracheophyta", no rank

"ferns and horsetails": "Moniliformopses", no rank

"lycophytes": " Lycopodiophyta", no rank

"seed plants": "Spermatophyta", no rank

"conifers": "Coniferophyta", no rank

"gymnosperms (conifers)": "Coniferophyta", no rank

"cycads": " Cycadophyta ", no rank

"ginkgos": "Ginkgophyta", no rank

"gnetophytes": "Gnetophyta", no rank

"flowering plants": "Magnoliophyta", no rank

The following comprise all immediate descendents of node "Magnoliophyta":

"basal magnoliophytes": "basal Magnoliophyta", no rank

"ceratophyllales": "Ceratophyllales", order

"eudicots": "eudicotyledons", no rank

"magnoliids": "magnoliids", no rank

"monocots": "Liliopsida", class

"unclassified magnoliophytes": "unclassified Magnoliophyta", no rank

The following are not monophyletic, but instead consist of multiple clades:

"bryophytes": "Anthocerotophyta" + "Marchantiophyta" + "Bryophyta"

"seed plants and ferns": any "Tracheophyta" NOT in clades "Moniliformopses", "Lycopodiophyta" or "Spermatophyta"

"ferns and allies": bryophytes (see above) + "Moniliformopses"

"gymnosperms (non-conifer)": "Cycadophyta " + "Ginkgophyta" + "Gnetophyta"

1. Prune the tree
	1. Discard all ancestors below the node subtending" Plants+Fungi". This becomes the root of the tree.
	2. Remove all taxa at rank lower than genus. This involves deleting all descendants of any taxon with rank='genus'
2. Add missing genera from the Tropicos & GRIN taxonomy
* Because the list of plant genera in NCBI is very incomplete, additional genera are added to tables clade and cladeName from Tropicos and GRIN Taxonomy.
* Tropicos genera and their families extracted from tnrs database, as extracted from Tropicos API (<http://services.tropicos.org/>)
* GRIN genera and their families from <http://www.ars-grin.gov/cgi-bin/npgs/html/index.pl>
* Darned if I know where the scripts are that I used to make these tables, but I have copied them over to BIEN2 database as: `tropicos\_genera` and `grin\_genera`. Please use them.
* Missing genera are added to table cladeName, joining by family. They are also added to cladeName. In both cases, `source` equals either "tropicos" or "grin".
1. Populate column higherPlantGroup in analytical database (bien\_web)
* This final step is performed by joining each BIEN taxon by genus to table clade and transferring the value in bien2.clade.higherPlantGroup to bien\_web.observation.higherPlantGroup
* This validation was performed only for the second generation analytical database (bien\_web). It was not used for the creation of bien2.viewFullOccurrence.
* Important: note that this update is done in the following two steps:

update bien\_web.observation c join bien2.clade d

on c.family=d.cladeNameStd

set

c.higherPlantGroup=d.higherPlantGroup

where

c.higherPlantGroup is null and

d.majorPlantGroup='Viridiplantae' and

d.rank='family';

update bien\_web.observation c join bien2.clade d

on c.genus=d.cladeNameStd

set

c.higherPlantGroup=d.higherPlantGroup

where

c.higherPlantGroup is null and

d.majorPlantGroup='Viridiplantae' and

d.rank='genus':

* Updating first by family minimizes anomalies due to homonym genera in different families.

## 13. Index threatened and endangered species

* This is not a separate validation step
* bien\_iucn table prepared by importing to MySQL the IUCN list of threatened and endangered plants, as obtained from <http://www.iucnredlist.org/initiatives/europe/publications>.
* Threatened status of species can be determined by joining to bien\_web.observation to bien2.iucn\_species on bien\_web.observation.taxon= bien2.iucn\_species.Species

## 14. Aggregate species abundances from plot observations

* This step was performed by Steve Dolin's students while constructing the table `PlotAggregateFact` in the core bien2 database. However, it was not done properly, so please follow the instructions below.
* The goal was to aggregate all countable observations of species from plots. Each species would be listed only once for each plot, followed by a count of the total number of individuals in each of the following classes (also show is their mapping to column names in bien2.viewFullOccurrence and bien\_web.):

|  |  |
| --- | --- |
| **Abundance class** | **Column name in analytical database** |
| All individuals | Abund |
| ≥1 cm dbh | Abund1 |
| ≥2.5 cm dbh | Abund2.5 |
| ≥10 cm dbh | Abund10 |

The last three categories should count individuals, not stems. Thus, a tree with three stem of the following size: 3.0 cm, 5.0 cm, 12 cm, would count as one in each of the categories abund, abund2.5, abund10, but would NOT count in the category abund1.

* This should result in the following relationship between size classes:

abund ≤ abund1 ≥ abund2.5 ≥ abund10

## 15. Standardize plot metadata regarding plot methodology

* This operation is performed only during the preparation of the second-generation analytical database, bien\_web.
* Script: bien\_web/observation.inc

Affected columns in table observation: plotCode, plotAreaHa, plotMinDbh, plotMethod

* All standardization are performed by the SQL statement shown in Appendix 4.

## 16. Index data providers

* Provides indexed searchable list of data providers for citations and acknowledgements
* This operation is performed only during the preparation of the second-generation analytical database, bien\_web.
* Involves parsing of the DataSource field in viewFullOccurrence, as multiple data providers are sometimes aggregated into single record
* Distinguishes between plot data source, specimen data source, and trait data source
* Should distinguish between aggregators and primary data providers
* Index data source names which are standard herbarium acronym
* Scripts:
	+ bien\_web/datasource.inc
	+ bien\_web/bienSources.sql
	+ bien\_web/sourceCorrected/[all scripts in this directory]
* Resulting tables in bien\_web analytical database:
	+ datasource
	+ dataSourceNameCorrected

## 17. Determine a "concensusHabit" (growth form) for each species

* Currently under construction
* Based on data in traits table (bien2.TraitObservation) and algorithms developed by Barbara Dobrin
* Critical for many analyses



Fig. 1. Relationship between working geovalidation table geoscrub in database geoscrub and source tables in database bien2. Every record in table geoscrub is derived either from a record in IndividualObservation or a record in PlotMetaDataDimension; the column geoscrub.sourceID thus links 1:1 either to a record in IndividualObservation or to a record in PlotMetaDataDimension, as determined by the value of geoscrub.sourceTable. Column geoscrub.sourceTable takes on one of two values: "IndividualObservation", "PlotMetaDataDimension". The primary key for table geoscrub, column geoscrubID, is an artificial autoincrement integer; however, columns sourceID and sourceType also comprise a candidate primary key (i.e., they are unique in combination).



Fig. 2. Geovalidation database `geoscrub` core schema. Table `geoscrub` contains locality information for all records in tables `IndividualObservation`,`SpecimenObservation` and `PlotMetaDataDimension` in database `bien2` (see Fig. 1).



Fig. 3. Auxillary tables involved in (a) detection of cultivated specimens (`herbaria`), (b) flagging FIA forest plots from disturbed sites (`PlotMetaDataDimension`, `FIA\_COND`) and (b) indexing New World countries (`newWorldCountries`) in table geoscrub. All tables are in database geoscrub, except for PlotMetaDataDimension, which is in the core bien2 database. `FIA\_COND` contains metadata on plot condition, and was obtained from the FIA website (<http://apps.fs.fed.us/fiadb-downloads/datamart.html>). See sections 5, 8, and 9 for details on the use of these tables.



Fig. 4. Relationship between geovalidation results tables (geoIndividualObservation and geoPlotMetaDataDimension), the original, source tables in the bien2 core database (ObservationSpecimen+IndividualObservation and PlotMetaDataDimension), and the working geovalidation table (geoscrub) in database geoscrub. The geovalidation results tables reside in database bien2, and hold the final results of geovalidation, as transferred from geoscrub.geoscrub during the final step in the geovalidation pipeline. Table geoIndividualObservation is linked 1:1 identifying with IndividualObservation; geoPlotMetaDataDimension is linked 1:1 identifying with PlotMetaDataDimension. Every record in table geoscrub is therefore derived either from a record in IndividualObservation or a records in PlotMetaDataDimension; the column geoscrub.sourceID links 1:1 either to a record in geoIndividualObservation or to a record in geoPlotMetaDataDimension. The table from which the record is derived is identified in the column geoscrub.sourceTable, which takes on one of two values ("IndividualObservation"," PlotMetaDataDimension"). The 1:1 optional links between geoscrub and geoIndividualObservation and geoscrub and geoPlotMetaDataDimension are also enforced by the same pair of candidate primary keys (sourceTable, sourceID). That is, a record in table geoscrub with sourceType="IndividualObservation" links to exactly one record in each of the tables geoIndividualObservation and IndividualObservation.



Fig. 5. Phylogenetic backbone tables `clade` and `cladeName` in the bien2 database are based on the tables `names` and `nodes` from the NCBI Taxonomy ftp site. See text for details.



Fig. 6. Tables of all unique genera, and their families, from the Tropicos and GRIN taxonomic database. Any genera not in NCBI are added to tables `clade` and `cladeName` from table `tropicos\_genera`, followed by any genera not in NCBI or Tropicos, from table `grin\_genera`.

Appendix 1. PHP functions for populating right and left indexes using modified pre-ordered tree traversal. get\_index() is the main (recursive) function. has\_children() and get\_children() are required functions. The main function is called by sending it the

function get\_index($tbl, $this\_id, $ind, $id\_name, $parent\_id\_name, $l\_ind\_name, $r\_ind\_name) {

 // Recursively increments left and right indices index via a modified tree-traversal algorithm

 // Initiate by sending id of root and ind=0

 // Sets left index and right index to NULL for all records on initial call (ie., when $ind=0)

 // This enable use of NULL check to detect circular references

 global $get\_index\_row, $get\_index\_tot\_rows, $get\_index\_start\_id;

 // reset both indices to null for all fields if this if initial call

 if ($ind==0) {

 $sql = "

 UPDATE `$tbl`

 SET `$l\_ind\_name`=NULL, `$r\_ind\_name`=NULL;

 ";

 if (!(mysql\_query($sql))) die("Failed to reset left and right indices for table `$tbl`.\r\n".mysql\_error().

 // get total records in table (for status message)

 $sql="

 SELECT `$id\_name` FROM `$tbl`;

 ";

 $result=mysql\_query($sql);

 $get\_index\_row=1;

 $get\_index\_tot\_rows = mysql\_num\_rows($result);

 $get\_index\_start\_id=$this\_id;

 }

 // display status message

 echo "\r Indexing row $get\_index\_row of $get\_index\_tot\_rows.";

 $get\_index\_row++;

 // perform NULL check for circular reference

 // left index should be NULL for this record

 $sql = "

 SELECT `$id\_name`

 FROM `$tbl`

 WHERE `$id\_name`=$this\_id

 AND (`$l\_ind\_name` IS NOT NULL);

 ";

 $result=mysql\_query($sql);

 $num\_rows = mysql\_num\_rows($result);

 if ($num\_rows>0) die("Error: circular reference at `$id\_name`=$this\_id.\r\n");

 // increment index and assign to left index

 $ind++;

 $l\_ind = $ind;

 // update left index for this node

 $sql = "

 UPDATE `$tbl`

 SET `$l\_ind\_name`=$l\_ind

 WHERE `$id\_name`=$this\_id;

 ";

 if (!(mysql\_query($sql))) die("Failed to update right index for $id\_name=$this\_id in table `$tbl`.\r\n".mysql\_error()."\r\n");

 // check if this node has children

 if (has\_children($this\_id,$tbl,$id\_name,$parent\_id\_name)) {

 // make array of child ids

 $child\_ids = get\_children($this\_id, $tbl, $id\_name, $parent\_id\_name);

 // request new index value from each child

 foreach ($child\_ids as $child\_id) {

 $ind = get\_index($tbl, $child\_id, $ind, $id\_name, $parent\_id\_name, $l\_ind\_name, $r\_ind\_name);

 }

 }

 // increment index again and assign to right index

 $ind++;

 $r\_ind = $ind;

 // update right index for this node and return new value of index

 $sql = "

 UPDATE `$tbl`

 SET `$r\_ind\_name`=$r\_ind

 WHERE `$id\_name`=$this\_id;

 ";

 if (!(mysql\_query($sql))) die("Failed to update right index for $id\_name=$this\_id in table `$tbl`.\r\n".mysql\_error()."\r\n");

 // If this is final call (back at root) print new line

 // Issue warning if not all records indexed

 if ($this\_id==$get\_index\_start\_id) {

 echo "\r\n";

 if ($get\_index\_row<$get\_index\_tot\_rows) echo " Warning: indexed rows < total rows in table.\r\n";

 }

 return $ind;

}

function get\_children($parent\_id, $tbl, $id\_name, $parent\_id\_name) {

 // compiles a unidimensional array consisting of PKs of all child records

 // table must contain a recursive FK linking children to parents

 $sql = "

 SELECT `$id\_name` AS child\_id

 FROM `$tbl`

 WHERE `$parent\_id\_name`=$parent\_id;

 ";

 if ($result=mysql\_query($sql)) {

 $num\_rows = mysql\_num\_rows($result);

 if ($num\_rows==0) {

 $children[]=0;

 } else {

 while ($row = mysql\_fetch\_array($result, MYSQL\_ASSOC)) {

 $children[] = $row{'child\_id'};

 }

 }

 }

 return $children;

}

function has\_children($this\_id, $tbl, $id\_name, $parent\_id\_name) {

 // returns true if one or more records link to this record via recursive FK

 $sql = "

 SELECT `$id\_name`

 FROM `$tbl`

 WHERE `$parent\_id\_name`=$this\_id;

 ";

 if ($result=mysql\_query($sql)) {

 $num\_rows = mysql\_num\_rows($result);

 if ($num\_rows>0) {

 return true;

 } else {

 return false;

 }

 } else {

 return false;

 }

}

Appendix 2. Example PHP script which adds root record to table $tbl and calls function get\_index() (see Appendix 1) to populate left and right indexes.

<?php

// generates right and left indices

// for hierarchical searches

if ($echo\_on) echo " Performing tree traversal:\r\n";

// Add root

// PK MUST be integer; modify if not

if ($add\_root) {

 echo " Adding root...";

 // get next value of PK

 $sql = "SELECT MAX($id\_fld) AS max\_id FROM $tbl;";

 $max\_id = sql\_get\_first\_result($sql,'max\_id');

 $root\_id = $max\_id + 1;

 // Add root record

 $sql="INSERT INTO $tbl(`$id\_fld`,`$name\_fld`,`$rank\_fld`)

 SELECT $root\_id, '$root\_name', '$root\_rank';";

 $msg\_error = "Failed to add root record!";

 if (sql\_execute($sql,$die\_on\_fail,$echo\_on,$msg\_success,$msg\_error));

}

// get root id and do some error checking

$sql = "SELECT `$id\_fld` FROM `$tbl` WHERE `$rank\_fld`='$root\_rank';";

if (!($result=mysql\_query($sql))) die("Failed to get id of root.\r\n");

$num\_rows = mysql\_num\_rows($result);

if ($num\_rows<1) {

 die("Error: no root");

} elseif ($num\_rows>1) {

 die("Error: more than one root");

} else {

 $row = mysql\_fetch\_row($result);

 $root\_id=$row[0];

}

if ($preroot) {

 // Add node above root for linking orphan taxa (no parentTaxonID)

 // Indexing algorithm may fail if have multiple taxa linked to root.

 echo " Adding pre-root...";

 // get next value of PK

 $sql = "SELECT MAX($id\_fld) AS max\_id FROM $tbl;";

 $max\_id = sql\_get\_first\_result($sql,'max\_id');

 $preroot\_id = $max\_id+1;

 // Add root record

 $sql="INSERT INTO $tbl(`$id\_fld`,`$parent\_id\_fld`,`$name\_fld`,`$rank\_fld`)

 SELECT $preroot\_id, $root\_id, 'preroot', 'preroot';";

 $msg\_error = "Failed to add preroot record!";

 if (sql\_execute($sql,$die\_on\_fail,$echo\_on,$msg\_success,$msg\_error));

 // Fix all NULL parentIDs by setting to prerootID

 // (except root itself --> must have NULL parentID)

 $sql = "UPDATE $tbl

 SET `$parent\_id\_fld`=$preroot\_id

 WHERE `$parent\_id\_fld` IS NULL AND $id\_fld<>$root\_id;";

 $msg\_error = "Failed to set NULL `$parent\_id\_fld` to PK of preroot!";

 if (sql\_execute($sql,$die\_on\_fail,$echo\_on,'',$msg\_error));

}

// Populate left and right indexes using recursive tree traversal

// Begins at root, traverses tree and returns to root

$ind = 0; // starting value of index

if (!($r\_ind=get\_index($tbl, $root\_id, $ind, $id\_fld, $parent\_id\_fld, $left\_index\_fld, $right\_index\_fld))) {

 echo " Indexing error!\r\n";

}

if ($preroot) {

 // Remove root and set preroot to root, and left index to 1

 // this means there will be no left index=2...c'est la vie

 echo " Removing pre-root...";

 // Delete root

 $sql="DELETE FROM $tbl

 WHERE `$id\_fld`=$root\_id;";

 $msg\_error = "Failed to delete temporary root!";

 if (sql\_execute($sql,$die\_on\_fail,$echo\_on,'',$msg\_error));

 // Reset preroot to root

 $sql="UPDATE $tbl

 SET `$rank\_fld`='$root\_rank', `$name\_fld`='$root\_name', `$parent\_id\_fld`=NULL, `$left\_index\_fld`=1

 WHERE `$id\_fld`=$preroot\_id;";

 $msg\_error = "Failed to reset preroot to root!";

 if (sql\_execute($sql,$die\_on\_fail,$echo\_on,$msg\_success,$msg\_error));

}

?>

Appendix 3. Examples of ancestor and descendent lookups using modified pre-ordered tree traversal indices

mysql> -- ancestor lookup to arbitrary depth using modified pre-ordered tree traversal indices

mysql> select rank, cladeNameStd

 -> from clade

 -> where

 -> leftIndex<=(select leftIndex from clade where cladeNameStd="Cycadophyta ")

 -> and

 -> rightIndex>=(select rightIndex from clade where cladeNameStd="Cycadophyta ")

 -> order by leftIndex;

+---------+----------------+

| rank | cladeNameStd |

+---------+----------------+

| root | Plants+Fungi |

| kingdom | Viridiplantae |

| phylum | Streptophyta |

| no rank | Streptophytina |

| no rank | Embryophyta |

| no rank | Tracheophyta |

| no rank | Euphyllophyta |

| no rank | Spermatophyta |

| no rank | Cycadophyta |

+---------+----------------+

9 rows in set (0.04 sec)

mysql> -- descendant lookup to arbitrary depth using modified pre-ordered tree traversal indices

mysql> select rank, cladeNameStd

 -> from clade

 -> where

 -> leftIndex>=(select leftIndex from clade where cladeNameStd="Cycadophyta ")

 -> and

 -> rightIndex<=(select rightIndex from clade where cladeNameStd="Cycadophyta ")

 -> order by leftIndex;

+---------+----------------+

| rank | cladeNameStd |

+---------+----------------+

| no rank | Cycadophyta |

| class | Cycadopsida |

| order | Cycadales |

| family | Zamiaceae |

| genus | Encephalartos |

| genus | Zamia |

| genus | Ceratozamia |

| genus | Dioon |

| genus | Macrozamia |

| genus | Chigua |

| genus | Microcycas |

| genus | Lepidozamia |

| genus | Palma-filix |

| genus | Palmifolium |

| genus | Catakidozamia |

| genus | Encephallartes |

| genus | Platyzamia |

| genus | Dion |

| genus | Katakidozamia |

| genus | Dipsacozamia |

| family | Cycadaceae |

| genus | Cycas |

| genus | Bowenia |

| genus | Epicycas |

| family | Stangeriaceae |

| genus | Stangeria |

+---------+----------------+

26 rows in set (0.00 sec)

mysql> -- retrieval of immediate descendants only using parent-child adjacency

mysql> select c.rank, c.cladeNameStd

 -> from clade as c join clade as p

 -> on c.parentCladeID=p.cladeID

 -> where p.cladeNameStd="Magnoliophyta";

+---------+----------------------------+

| rank | cladeNameStd |

+---------+----------------------------+

| class | Liliopsida |

| no rank | eudicotyledons |

| order | Ceratophyllales |

| no rank | magnoliids |

| no rank | basal Magnoliophyta |

| no rank | unclassified Magnoliophyta |

+---------+----------------------------+

6 rows in set (0.00 sec)

Appendix 4. SQL used to standardize plot metadata fields pertaining to methodology in table bien\_web.observation.

 update bien\_web.observation c join bien2.PlotMetaDataDimension p

 on c.bien2\_DBPlotID=p.DBPlotID

 set

 c.plotCode=if(DBSourceName='FIA',p.PlotCD,c.plotCode),

 c.plotAreaHa=if(p.PlotArea+0.0=0,null,p.PlotArea+0.0),

 c.plotMinDbh=if(p.PlotMinDbh=10.00000, 10,

 if(p.PlotCensusMethod like '%2.5 c%',2.5,

 if(p.PlotCensusMethod like '%10 cm%' or p.PlotCensusMethod like '%10cm%',10,null)

 )

 ),

 c.plotMethod=if(p.PlotCensusMethod like '%transect%','0.1 ha transect',

 if(p.PlotCensusMethod like '1 ha%','1 ha plot',

 if(p.PlotCensusMethod like 'Point-intercept%', 'Point-intercept',NULL)

 )

 )

 where c.bien2\_DBPlotID is not null;