processALA: An R-package to help manage ALA data for plant species

Peter D. Wilson

21 January 2023

processALA is an ***R***-package developed to make downloading and cleaning occurrence data for plant species from the Atlas of Living Australia (ALA) a simple process. By stream-lining the way you interact with ALA, the package should allow you to check taxonomic names and obtain useful occurrence data "live". That is, you can integrate the process of downloading the latest occurrence data into *R*-scripts and run them whenever you need up-to-date data. Alternatively, you can use the functions included in the package to run data download and cleaning to periodically refresh sets of local stored data files. This second approach can be the most efficient method because data from ALA changes infrequently, which is particularly true of herbarium data which is the highest value data for most use cases we encounter. In addition, live interrogation of ALA depends on a good internet connection and naturally introduces delays to running scripts.

A major re-working of **processALA** was undertaken in late November 2021 to transition from using the package **ALA4R** to the package **galah**. **ALA4R** was retired at the end of 2021 in favour of its replacement package **galah**. Development of **processALA** has been ongoing since then because the developers of **galah** frequently make significant and unannounced changes to critical data structures and functions within the package **galah**. The changes to the **galah** package reflect significant and unannounced changes to critical data structure in the ALA infrastructure.

To view and download the *latest* source for **processALA** go to: https://github.com/peterbat1/processALA.

To install the package in your local ***R*** installation, it is easiest to ensure that the following dependencies are installed in your local ***R*** system:

- galah
- httr
- dplyr
- ozmaps (for running the worked example in this vignette)

It is most productive to install these packages using the package management features in *R-Studio* as they pull in a fair number of dependencies and it is a pain to work through that list manually before you finally install the target package.

To complete the installation of **processALA**, ensure that the package **remotes** is installed from CRAN, then enter the following command at the prompt in the ***R*** console:

```
remotes::install_github("peterbat1/processALA", build_vignettes = TRUE)
```

As with any ***R***-package installed on your system, your scripts must begin with:

```
library(processALA)
```

A bit about ALA and accessing data

The Atlas of Living Australia is many things:

- A gateway to a comprehensive collection of occurrence records of many types including herbarium records from Australia and New Zealand supplied by Australia's Virtual Herbarium (which is hosted by ALA)
- The source of an accepted National Species List, and repository of taxonomic and nomenclatural information which, for plants, is formed by the Australian Plant Name Index (APNI) and the Australian Plant Census (APC) hosted by ALA on behalf of the Council of Heads of Australasian Herbaria (CHAH)
- The Australian node of the Global Biodiversity Information Facility (GBIF)

> **NOTE:** ALA seems to have episodes where there is a long delay between a change in a taxonomic name and the re-grouping of occurrence records under the new name. Thankfully, this affects only a handfull of taxa, but delays can be inexplicably long. I have found instances where the delay in propagating name changes has been 4 years! A current example of this situation (as at 2023-07-05) is *Desmodium campylocaulon* Creeping Tick-Trefoil which was replaced by *Desmodiopsis campylocaulon* in 2018. Searching ALA under the old name does not refer you to the new name, and instead returns 375 occurrence records. Searching under the new name gives 44 records totaly unrelated to those found using the old name. Support from ALA says we will have to manually merge the occurrence records until ALA sorts their indexing and cross-linking out…hopefully in less than another 4 years! Note that searching APNI or APC directly correctly links the two names (i.e. *Desmodium campylocaulon* is shown as a homotypic synomym of *Desmodiopsis campylocaulon*), but you first have to know that there is a glitch in retrieving occurrence records to deal with the consequences of the lag between APNI/APC and ALA occurrence retrieval processes.

ALA is an important source of occurrence data and taxonomic information which can be accessed through a browser-based web interface. However, for many researchers this can be cumbersome, particularly when many taxa must be processed. Fortunately, ALA also provides an API or Application Programming Interface which lets you perform searches and retrieve records using scripts. This makes it easier to process lists of taxa or make periodic updates of information. You can find out more about the ALA API here.

ALA also maintains an ***R***-package, **galah**, to make it easy to use the ALA API. The functions within that package allow us to make use of the API without having to learn the ins and outs of composing query strings, and the application of web access tools such as *curl* and *wget*. (Of course, if you're into such things there are fun times ahead if you want to work directly with curl or wget calls.)

The package **processALA** adds another level of refinement. It highly automates key steps in the retrieval and processing of ALA data by utilising a few key functions in the **galah** package and adds functions to assist in data cleaning. In addition, it adds a mechanism to access some ALA API functions not exposed in **galah**.

> **NOTE:** You must register an email address with ALA before you can use **processALA** to download occurrence data. Once you have done this, you should enter the following line in to the R console: galah::galah_config(email = "your_email@blah.com") obviously substituting the registered email address for "your_email@blah.com". This will ensure that, when functions in **processALA** call functions in **galah**, you will be recognised by ALA and the data you request will be supplied.
>
> To create an account with ALA, use this link: https://auth.ala.org.au/userdetails/

registration/createAccount

There are three stages to obtaining occurrence data from ALA:

- resolving your taxa to those available on the National Species List
- downloading occurrence data
- cleaning occurrence data

Three key functions in **processALA** have been designed to implement each step. The functions are:

- *checkTaxonName()*
- *fetchALAdata()* and
- *filterALAdata()*

The focus of this vignette is to guide you in applying them to obtaining taxonomic information, fetching occurrence records and making them fit for further use.

Handling taxonomic names

The ALA API provides a number of functions which allow you to interrogate APNI and APC to resolve plant names to a matching accepted name from APC. A function in **galah**, *select_taxa()* is available to determine the status of a name and provide a few details associated with it. The **processALA** function *checkTaxonName()* uses that function to check a name you have supplied and also makes a few calls to ALA API functions not accessible via **galah**. The returned information is useful for further processing of records, and for creating or updating a taxonomic table for your work.

Steps to checking a name include:

1. Call *checkTaxonName()* with your prospective name as the first parameter and assign the result to an object
2. Examine data elements in the result object and use them to determine the next step

For example:

```
ans <- checkTaxonName("Acacia linifolia")
ans
#>   isValid isAccepted       searchName searchName_taxonomicStatus
#> 1    TRUE       TRUE Acacia linifolia                   accepted
#>       acceptedName                fullAcceptedName  genus specificEpithet
#> 1 Acacia linifolia Acacia linifolia (Vent.) Willd. Acacia       linifolia
#>   infraSpecificRank infraSpecificEpithet infraSpecificRankAbbrev    taxonAuthor
#> 1                                                                (Vent.) Willd.
#>   acceptedGUID                                acceptedFullGUID
#> 1      2906316 https://id.biodiversity.org.au/node/apni/2906316
#>                  formattedAcceptedName taxonomicStatus totalRecords
#> 1 <i>Acacia linifolia</i> (Vent.) Willd.        accepted         4967
#>   taxonomicRank parentGUID parentName parentTaxonomicRank
#> 1       species   51382879     Acacia               genus
#>
      synonyms
#> 1 Acacia linearis; Acacia linearis; Acacia linifolia var. linifolia; Mimosa
      abietina; Mimosa linearis; Phyllodoce linifolia; Racosperma linifolium; Mimosa
      linifolia; Acacia abietina
```

```
#>   apcFamily
#> 1  Fabaceae
#>
        alaCommonNames
#> 1 Flax-Leaved Acacia; Flax-Leaved Wattle; Flax-Leaved Wattle; Flax-Leaved Wattle;
        White Wattle; White Wattle
#>   inferredAcceptedInfo
#> 1
```

The result object **ans** is a 1-row data frame with the following columns or data elements:

Element	Data type and contents
isValid	Logical. Is the taxonomic name found in the Australian Plant Name Index (APNI); to be listed in APNI a taxonomic name must have been formally defined and applied in a publication to collected material.
isAccepted	Logical. Is the searched for taxonomic name accepted by the Australian Plant Census (APC) as an accepted name associated with a taxonomic concept.
searchName	Character (string). Taxonomic name searched for.
searchName_taxonomicStatus	Character (string). Taxonomic status of the name you searched on. This may be: “Accepted”, “Inferred accepted”, “Not accepted”, “Synonym”.
acceptedName	Character (string). The accepted taxon name corresponding to the name searched for.
fullAcceptedName	Character (string). Binomial/trinomial + author of the accepted taxon name.
acceptedGUID	Character (string). The ALA Globally Unique Identifier or GUID; a numeric string uniquely indexing a taxonomic concept in the Australian Plant Name Index (APNI) component of the National Species List (NSL). No match results in a GUID of “Not_accepted” being returned.
acceptedFullGUID	Character (string). The full URL form of the GUID: stored as a convenience to speed calls to the ALA API, and assist maintenance of code.
formattedAcceptedName	Character (string). Full taxonomic name (i.e. binomial/trinomial + author) with simple HTML mark-up to italicise the binomial/trinomial part.
taxonomicRank	Character (string). Taxonomic rank of the accepted name.
parentGUID	Character (string). Full GUID of the taxonomic parent of the accepted taxon.
parentName	Character (string). Name of the taxonomic parent of the accepted taxon.
parentTaxonomicRank	Character (string). Taxonomic rank of the parent.
synonyms	Character (string). A semi-colon separated list of taxonomic synonyms of the accepted taxon.
apcFamily	Character (string). Family of the accepted taxon according to APC.

Element	Data type and contents
alaCommonNames	Character (string). Shows any common names listed by ALA.
inferredAcceptedInfo	Character (string). Shows Name, Source of information and Common names for all taxa associated with the search name or synonyms of the accepted taxon concept.

A name is "valid" in the sense used for **processALA** when it is found in an APNI record. However, the name you search on may not be the name which is currently applied to the taxon according to APC. APC lists *formally accepted* taxonomic names whereas APNI lists all published names associated with the currently accepted name listed on APC. Thus, we can be in one of three states after a call to *checkTaxonName()* based on the logical flags *isValid* and *isAccepted*:

isValid	isAccepted	Meaning
FALSE	FALSE	You have found nothing matching any name listed in APNI and, naturally, there is no corresponding accepted name in APC.
TRUE	FALSE	Your search name is listed in APNI but it is not the accepted name for the taxon according to APC. The corresponding APC accepted name will be shown in the *acceptedName* column of the result. Your search name will be shown as a synonym of the accepted name in the field *synonyms*.
TRUE	TRUE	Congratulations! The name you searched for is listed in APNI and is the accepted name according to APC. In this case, *searchName* and *acceptedName* will be identical.

Here are some quick examples of calls to *checkTaxonName()* resulting in each state:

FALSE : FALSE

```
ans <- checkTaxonName("Greenus plantus")
ans
#>   isValid isAccepted      searchName searchName_taxonomicStatus acceptedName
#> 1    TRUE      FALSE Greenus plantus                   accepted    Stylidium
#>          fullAcceptedName     genus specificEpithet infraSpecificRank
#> 1 Stylidium Sw. ex Willd. Stylidium
#>   infraSpecificEpithet infraSpecificRankAbbrev   taxonAuthor acceptedGUID
#> 1                                               Sw. ex Willd.     51311312
#>                                      acceptedFullGUID
#> 1 https://id.biodiversity.org.au/taxon/apni/51311312
#>                         formattedAcceptedName taxonomicStatus totalRecords
#> 1 <i>Stylidium</i> Stylidium Sw. ex Willd.        accepted        66617
#>   taxonomicRank parentGUID   parentName parentTaxonomicRank
#> 1         genus   51311311 Stylidiaceae              family
#>
      synonyms
#> 1 Stylidium; Ventenatia; Candollea; Forsteropsis; Oreostylidium; Stylidium ser.
      Imbricatae
#>      apcFamily
#> 1 Stylidiaceae
#>
      alaCommonNames
#> 1 Hair-Trigger; Spring Back; Spring-Back Plant; Trigger Flower; Trigger Plants;
      Trigger Plants; Trigger-Plants; Triggerplants
#>   inferredAcceptedInfo
#> 1
```

TRUE : FALSE

```
ans <- checkTaxonName("Acacia abietina")
ans
#>   isValid isAccepted      searchName searchName_taxonomicStatus
#> 1    TRUE      FALSE Acacia abietina         heterotypicSynonym
#>       acceptedName                fullAcceptedName  genus specificEpithet
#> 1 Acacia linifolia Acacia linifolia (Vent.) Willd. Acacia       linifolia
#>   infraSpecificRank infraSpecificEpithet infraSpecificRankAbbrev    taxonAuthor
#> 1                                                                (Vent.) Willd.
#>   acceptedGUID                                 acceptedFullGUID
#> 1      2906316 https://id.biodiversity.org.au/node/apni/2906316
#>                    formattedAcceptedName taxonomicStatus totalRecords
#> 1 <i>Acacia linifolia</i> (Vent.) Willd.        accepted       876053
#>   taxonomicRank parentGUID parentName parentTaxonomicRank
#> 1       species   51382879     Acacia               genus
#>
      synonyms
#> 1 Acacia linearis; Acacia linearis; Acacia linifolia var. linifolia; Mimosa
      abietina; Mimosa linearis; Phyllodoce linifolia; Racosperma linifolium; Mimosa
      linifolia; Acacia abietina
#>   apcFamily
#> 1  Fabaceae
#>
      alaCommonNames
#> 1 Flax-Leaved Acacia; Flax-Leaved Wattle; Flax-Leaved Wattle; Flax-Leaved Wattle;
      White Wattle; White Wattle
#>   inferredAcceptedInfo
#> 1
```

TRUE : TRUE

```
ans <- checkTaxonName("Acacia linifolia")
ans
#>   isValid isAccepted       searchName searchName_taxonomicStatus
#> 1    TRUE       TRUE Acacia linifolia                   accepted
#>       acceptedName                fullAcceptedName  genus specificEpithet
#> 1 Acacia linifolia Acacia linifolia (Vent.) Willd. Acacia       linifolia
#>   infraSpecificRank infraSpecificEpithet infraSpecificRankAbbrev    taxonAuthor
#> 1                                                               (Vent.) Willd.
#>   acceptedGUID                                 acceptedFullGUID
#> 1      2906316 https://id.biodiversity.org.au/node/apni/2906316
#>                     formattedAcceptedName taxonomicStatus totalRecords
#> 1 <i>Acacia linifolia</i> (Vent.) Willd.        accepted         4967
#>   taxonomicRank parentGUID parentName parentTaxonomicRank
#> 1       species   51382879     Acacia               genus
#>
      synonyms
#> 1 Acacia linearis; Acacia linearis; Acacia linifolia var. linifolia; Mimosa
      abietina; Mimosa linearis; Phyllodoce linifolia; Racosperma linifolium; Mimosa
      linifolia; Acacia abietina
#>   apcFamily
#> 1  Fabaceae
#>
      alaCommonNames
#> 1 Flax-Leaved Acacia; Flax-Leaved Wattle; Flax-Leaved Wattle; Flax-Leaved Wattle;
      White Wattle; White Wattle
#>   inferredAcceptedInfo
#> 1
```

Hopefully you can see the potential for using the information returned by *checkTaxonName()*. For example:

- You should be able to write some ***R*** code to flag bad names and have your script act accordingly.
- You can easily add a list of synonyms to your output files or documents by using text stored in the *synonyms* field of the object returned by *checkTaxonName()*.
- You can readily insert the full accepted taxonomic names into output, including using the HTML-formatted version in web-based outputs (e.g. ***R*** Shiny apps).

Downloading occurrence data

The next step is downloading occurrence data for an accepted taxonomic name. This is performed by the function *fetchALAdata()* which uses the function *ala_occurrences()* in the package **galah**. Note that the parameter *doNameCheck* is by default set to **TRUE** so that a name check is performed before attempting a data download. However, if a prior call has been made to *checkTaxonName()*, then this parameter may be set to **FALSE** and so some small quantum of time may be saved.

The **galah** function *ala_occurrences()* returns a data frame which is the table of occurrence data. *fetchALAdata()* saves the downloaded data in the specified output folder using the following naming convention: “Genus_specificEpithet.csv”.

Within the raw data table maybe found occurrence records for several record types which are indicated in the field *basisOfRecord*. The record types include the following:

basisOfRecord	Interpretation
PreservedSpecimen	Herbarium records passed into ALA from the consortium of herbaria and museums which supply data to Australasian Virtual Herbarium (AVH) and which is accessed via ALA

basisOfRecord	Interpretation
HumanObservation	Human observations which means any record of species occurrence arising from field encounters or observations which did not result in a voucher specimen being lodged in an herbarium. Examples include casual observations by the general public (also sometimes referred to as 'incidental observations'), and data from systematic vegetation surveys.
Image	An image of a species which is supplied with geo-coordinates. There are typically few of these encountered in most data downloads.
LivingSpecimen	A form of human observation or incidental observation referring to a specimen in a living collection such as a botanic garden or arboretum.
	A small fraction of records have the basisOfRecord field empty!

The downloaded occurrence data represents **raw** ALA occurrence data which must be cleaned or filtered before use. However, one preliminary step which *fetchALAdata()* performs is to split three data types out from the raw data table and place them into separate csv-formatted files.

The first is the *PreservedSpecimen* records which are saved in a file named "Genus_species_herbariumRecords.csv". Second, *HumanObservation* records are placed in a similarly named file, "Genus_specificEpithet_humanObservations.csv". Finally, *HumanObservation* records are parsed to identify records generated by surveys forming part of the NSW Vegetation Information System (VIS). If any are found, they are saved as "Genus_specificEpithet_surveyRecords.csv". Note that these data will also be present in the human observation data.

As a final point, note that image and living specimen data are not currently extracted and saved. These data are however present in the raw data csv file and you may therefore use an ***R***-script or spreadsheet program (Excel, or LibreOffice Calc on Linux PCs) to extract them if you need access to them.

Filtering/cleaning ALA data

ALA data is ***dirty*** data! ALA is just a data aggregation service and web portal transmitting the data as received from data providers. ALA does perform data quality checks and flags as many possible errors as it can, but the data remain as supplied and may include a range of errors which must be addressed before that data can be used with confidence. It is up to the end users to either use the quality flags raised by ALA or, preferably, do additional checks to arrive at the cleanest occurrence data possible.

> **It is absolutely essential that you perform data cleaning or filtering before using ALA occurrence data.**

Accumulated experience indicates that there are three types of error present in ALA occurrence data. These include:

- Location errors
- Taxon ID errors

- Record type errors

These three types of error have been identified in many other aggregated biodiversity data sets including the Global Biodiversity Information Facility (GBIF). We will take a quick look at each error type and then see how the function *filterALAdata()* attempts to remove each of them from raw ALA occurrence data.

Location errors

Location errors are caused by several conditions:

- missing geo-coordinates (lat/longs)
- inaccurate geo-coordinates: poor data recording by field collectors, data entry errors and, for very old records, imprecise location descriptions which cannot be interpreted to derive meaningful geo-coordinates

Location accuracy and record completeness has improved greatly in recent years and is particularly good from the late 1990s with the increasing use of GPS technology. There is clearly a temporal dimension to geo-coordinate data quality - older records have higher missing coordinate rates and lower accuracy.

filterALAdata() filters missing coordinates by default (i.e. the parameter *removeMissingCoordinates* = TRUE by default).

A further level of filtering location errors is achieved by calling *filterALAdata()* with the parameter *filterByJurisdiction* = TRUE (which is the default). When set, this will cause *filterALAdata()* to use the **processALA** function *fetchJurisdictionInfo()* to ask ALA for a list of Australian states and territories in which the taxon is found. Records which fall in a state or territory which is *not* expected are removed. Any record falling outside Australia (either because it is an herbarium specimen in an Australian herbarium collection but collected outside Australia, or because of a gross coordinate error leading to mid-ocean specimens of terrestrial species!) is also removed in this process.

A list of the states and territories included in *fetchJurisdictionInfo()*, and therefore used by *filterALAdata()*, is provided in the help page for *fetchJurisdictionInfo()*.

Taxon ID errors

Taxonomic errors may arise from mis-identification in the field which is subsequently not corrected. This type of error can be extremely hard to identify in ALA raw data. In some instances such errors may be apparent when there are a few spatial outliers beyond the expected distribution of a taxon. However, such spatial outliers can also be due to bad geo-coordinates or record type errors.

ALA uses the resources APNI and APC to try and resolve taxonomic names supplied by data providers and in most instances this is highly successful. However, this error-trapping cannot be guaranteed to be 100% effective. Whenever it is feasible, you should review downloaded to look for any anomalies.

There is also a temporal dimension to taxonomic name errors. Very old specimens may have undergone many complex taxonomic revisions which may make it difficult to assign a modern taxonomic concept to a specimen. This is particularly so when there have been taxonomic splits, and even more problematic when there have been splits for sympatric forms. It the last case, it may be impossible to fix taxonomic problems and you will have to make a call on what to do with taxa in this category and their occurrence data.

filterALAdata() cannot deal directly with these residual taxonomic ID errors but the parameter *filterByJurisdiction* will move some of them.

Record-type errors

There are relatively few occurrence records in ALA raw data with bad record types but they do occur and typically generate weird spatial outliers. In my experience, the most apparent record type error is caused by herbarium specimens from cultivated plants which are not correctly identified by ALA's automatic methods.

ALA sets a flag in the database field *occCultivatedEscapee* which appears to flow from a flag set by the AVH data aggregation process. However, a few data providers seem to be lax in their marking of herbarium specimens taken from cultivated individuals - Canberra and Melbourne are the key offenders - and this requires a second pass over the data by the end user. This additional filter is supplied by *filterALAdata()* by setting the parameter *filterCultivated* to TRUE. This is the default setting for the function and removes any record where the term "CULTIVATED" appears in the *locality* field.

As noted, rogue herbarium specimens from cultivated individuals often show up as inexplicable extreme spatial outliers. This also frequently means they are removed by the jurisdication filter applied by default when you call *filterALAdata()*. Experience suggests that the combined effect of applying *filterCultivated* = TRUE and *filterByJurisdiction* = TRUE removes nearly all herbarium samples collected from cultivated plants.

Worked example

Here is a short script which can be a starting point for developing your own workflows. It first calls *checkTaxonName()* to determine the status of the species name. Using this result, we determine which of the three states the result represents by comparing the flags *isValid* and *isAccepted*, and taking appropriate action.

> **Note that this script may take a minute or two to run as it fetches the occurrence data *live* from ALA.**

```
library(ozmaps)

# Example of an accepted name
thisTaxon <- "Alectryon coriaceus"

# Example of a synonym which is resolved to the accepted name Callistemon purpurascens
        because isValid == TRUE but isAccepted == FALSE
# thisTaxon <- "Callistemon sp. Purpurascens (S.Douglas s.n., 15 Dec. 2010)"

# A nonsense name as an example of the final state: isValid == FALSE and isAccepted ==
        FALSE
# thisTaxon <- "Greenus plantus"

cat("Checking", thisTaxon, "with ALA\n")
nameResult <- checkTaxonName(thisTaxon)

if ((nameResult$isValid) && (nameResult$isAccepted))
{
  cat(thisTaxon, "is valid and accepted\n")
  # Taxon name is valid and accepted so full steam ahead...
  fetchALAdata(thisTaxon, doNameCheck = FALSE)
```

```
    # Apply the default filter set on the downloaded data...
    filterALAdata(thisTaxon, recType = "herbarium", doNameCheck = FALSE)
  } else
  {
    if (nameResult$isValid)
    {
      thisTaxon <- nameResult$acceptedName
      # Name is present in APNI but is a synonym
      cat("ALA says the accepted name for", thisTaxon, "is", nameResult$acceptedName,
          "\n")
      fetchALAdata(nameResult$acceptedName, doNameCheck = FALSE)
      filterALAdata(nameResult$acceptedName, recType = "herbarium", doNameCheck = FALSE)
    }
    else
    {
      cat(thisTaxon, "is gibberish according to ALA - check, correct and try again\n")
    }
  }

  herbDataFile <- paste0(defaultOutputFolder, "/", thisTaxon, "/", gsub(" ", "_",
          thisTaxon, fixed = TRUE), "_herbariumRecords_filtered.csv")

  if (file.exists(herbDataFile))
  {
    # Make a map
    d <- read.csv(herbDataFile)

    oz()
    points(d$longitude, d$latitude, pch = 16, col = "darkorange")
  }
```