Beyond your first ticket

This section is meant to be read after you have processed a couple of tickets and you are comfortable with the workflow with a relatively simple dataset with 1-2 files and want to expand your skills and workflows further.

Working with large data packages

Add physicals to submissions

New submissions made through the web editor will not have any `physical` sections within the `otherEntitys`. Add them to the EML with the following script:

```
for (i in seq_along(doc$dataset$otherEntity)) {
    otherEntity <- doc$dataset$otherEntity[[i]]
    id <- otherEntity$id

    if (!grepl("urn-uuid-", id)) {
        warning("otherEntity ", i, " is not a pid")

    } else {
        id <- gsub("urn-uuid-", "urn:uuid:", id)
        physical <- arcticdatautils::pid_to_eml_physical(mn, id)
        doc$dataset$otherEntity[[i]]$physical <- physical
    }
}
```

As you can see from code above, we use a for loop here to add `physical` sections. The for loop is a very useful tool to iterate over a list of elements. With for loop, you can repeat a specific block of code without copying and pasting the code over and over again. When processing datasets in Arctic Data Center, there are many places where for loop can be used, such as publishing a bunch of objects with pids, updating formatID for pkg$data, adding `physical` section like above code, etc.

A loop is composed of two parts: the sequence and the body. The sequence usually generates indices to locate elements and the body contains the code that you want to iterate for each element.

Here is an example of adding the same `attributeList` for all the `dataTables` in the metadata using for loop.

```
attributes <- read.csv('attributes.csv')  # attribute table in csv format
attributeList <- EML::set_attributes(attributes = attributes)
```

```r
for (i in 1:length(doc$dataset$dataTable)) { # sequence part
    doc$dataset$dataTable[[i]]$attributeList <- attributeList # body part
}
```

Use references

References are a way to avoid repeating the same information multiple times in the same EML record. There are a few benefits to doing this, including:

- Making it clear that two things are the same (e.g., the creator is the same person as the contact, two entities have the exact same attributes)
- Reducing the size on disk of EML records with highly redundant information
- Faster read/write/validate with the R EML package

You may want to use EML references if you have the following scenarios (not exhaustive):

- One person has multiple roles in the dataset (creator, contact, etc)
- One or more entities shares all or some attributes

Example with parties

> **Note**
>
> Do not use references for creators as it is used for the citation information. The creators will not show up on the top of the dataset if it is a reference. Until this issue is resolved in NCEAS/metacat#926 we will need to keep this in account.

It's very common to see the contact and creator referring to the same person with XML like this:

```
<eml packageId="my_test_doc" system="my_system" xsi:schemaLocation="eml://ecoinformatics.org/
  <dataset>
    <creator>
      <individualName>
        <givenName>Bryce</givenName>
        <surName>Mecum</surName>
      </individualName>
    </creator>
    <contact>
      <individualName>
```

```
        <givenName>Bryce</givenName>
        <surName>Mecum</surName>
      </individualName>
    </contact>
  </dataset>
</eml>
```

So you see those two times Bryce Mecum is referenced there? If you mean to state that Bryce Mecum is the creator and contact for the dataset, this is a good start. But with just a name, there's some ambiguity as to whether the creator and contact are truly the same person. Using references, we can remove all doubt.

```
doc$dataset$creator[[1]]$id  <- "reference_id"
doc$dataset$contact <- list(references = "reference_id")
print(doc)
```

```
<?xml version="1.0" encoding="UTF-8"?>
<eml:eml xmlns:eml="eml://ecoinformatics.org/eml-2.1.1" xmlns:xsi="http://www.w3.org/2001/XM
  <dataset>
    <title>A Minimal Valid EML Dataset</title>
    <creator id="reference_id">
      <individualName>
        <givenName>Bryce</givenName>
        <surName>Mecum</surName>
      </individualName>
    </creator>
    <contact>
      <references>reference_id</references>
    </contact>
  </dataset>
</eml:eml>
```

> **Note**
>
> The reference `id` needs to be unique within the EML record but doesn't need to have meaning outside of that.

Example with attributes

To use references with attributes:

1. Add an attribute list to a data table
2. Add a reference id for that attribute list
3. Use references to add that information into the `attributeLists` of the other data tables

For example, if all the data tables in our data package have the same attributes, we can set the attribute list for the first one, and use references for the rest:

```
doc$dataset$dataTable[[1]]$attributeList <- attribute_list
doc$dataset$dataTable[[1]]$attributeList$id <- "shared_attributes" # use any unique name f

for (i in 2:length(doc$dataset$dataTable)) {
  doc$dataset$dataTable[[i]]$attributeList <- list(references = "shared_attributes") # use
}
```

Annotations

If there are multiple tables with similar annotations you can try something like this:

```
#go through each dataTable
for(i in 1:length(doc$dataset$dataTable)){

  #get all the attibute names
  an <- eml_get_simple(doc$dataset$dataTable[[i]], "attributeName")

  #go through the attributes figure out what will match
  for(a in 1:length(an)){
    annotation <- dplyr::case_when(
      # the attributeName to match ~ valueLabel
      an[[a]] == "Sample ID" ~ "Identity",
      an[[a]] == "Location" ~ "study location name",
      an[[a]] == "Latitude" ~ "latitude coordinate",
      an[[a]] == "Longitude" ~ "longitude coordinate",
      an[[a]] == "Elevation (m)" ~ "elevation",
      str_detect(an[[a]], "Depth|depth") ~ "Depth")

    #only run this code when the annotations match
    if(!is.na(annotation)){
      #based on the entity Name create a unique id
      entity <- str_split(doc$dataset$dataTable[[i]]$entityName, "_")
      doc$dataset$dataTable[[i]]$attributeList$attribute[[a]]$id <- paste0(entity[[1]][[1]
```

```
      #add the annotation
      doc$dataset$dataTable[[i]]$attributeList$attribute[[a]]$annotation <- eml_ecso_annot
    }
  }

}
```

Streamlining your workflow

Code Snippets

Code snippets help with templating portions of code that you will be using regularly. To add your own, go to the toolbar ribbon at the top of your Rstudio screen and select:

`Tools > Global Options... > Code > Edit Snippets > Add these chunks to the end of the file`

More info can be found in this blog post by Mara Averick on how to add them: https://maraaverick.rbind.io/2017/09/custom-snippets-in-rstudio-faster-tweet-chunks-for-all/

Usual `arcticdatautils` ticket workflow

```
snippet ticket
    library(dataone)
    library(arcticdatautils)
    library(EML)

    cn <- CNode('PROD')
    adc <- getMNode(cn, 'urn:node:ARCTIC')

    rm <- "add your rm"
    pkg <- get_package(adc, rm)
    doc <- EML::read_eml(getObject(adc, pkg\$metadata))

    doc <- eml_add_publisher(doc)
    doc <- eml_add_entity_system(doc)

    eml_validate(doc)
    eml_path <- "eml.xml"
    write_eml(doc, eml_path)
```

```
    #update <- publish_update(adc,
    #                                                metadata_pid = pkg\$metadata,
    #                                                resource_map_pid = pkg\$resource_map,
    #                                                metadata_path = eml_path,
    #                                                data_pids = pkg\$data,
    #                                                public = F)

    #datamgmt::categorize_dataset(update\$metadata, c("theme1"), "Your Name")
```

The `datapack` ticket workflow

```
snippet datapack
    library(dataone)
    library(datapack)
    library(digest)
    library(uuid)
    library(dataone)
    library(arcticdatautils)
    library(EML)

    d1c <- D1Client("PROD", "urn:node:ARCTIC")
    packageId <- "id here"
    dp <- getDataPackage(d1c, identifier=packageId, lazyLoad=TRUE, quiet=FALSE)

    #get metadata id
    metadataId <- selectMember(dp, name="sysmeta@formatId", value="https://eml.ecoinformat

    #edit the metadata
    doc <- read_eml(getObject(d1c@mn, metadataId))

    #add the publisher info
    doc <- eml_add_publisher(doc)
    doc <- eml_add_entity_system(doc)

    doc\$dataset\$project <- eml_nsf_to_project("nsf id here")

    #check and save the metadata
    eml_validate(doc)
    eml_path <- arcticdatautils::title_to_file_name(doc\$dataset\$title)
    write_eml(doc, eml_path)
```

```
    dp <- replaceMember(dp, metadataId, replacement=eml_path)

    #upload the dataset
    myAccessRules <- data.frame(subject="CN=arctic-data-admins,DC=dataone,DC=org", permiss
    packageId <- uploadDataPackage(d1c, dp, public=F, accessRules=myAccessRules, quiet=FAL
    #datamgmt::categorize_dataset("doi", c("theme1"), "Jasmine")
```

Quick way to give access to submitters to their datasets:

```
snippet access
    library(dataone)
    library(arcticdatautils)
    library(EML)

    cn <- CNode('PROD')
    adc <- getMNode(cn, 'urn:node:ARCTIC')

    rm <- "rm here"
    pkg <- get_package(adc, rm)

    set_access(adc, unlist(pkg), "orcid here")
```

Quick access to the usual code for common Solr queries:

```
snippet solr
    library(dataone)
    library(arcticdatautils)
    library(EML)

    cn <- CNode('PROD')
    adc <- getMNode(cn, 'urn:node:ARCTIC')

    result <- query(adc, list(q = "rightsHolder:*orcid.org/0000-000X-XXXX-XXXX* AND (*:* N
                              fl = "identifier,rightsHolder,formatId, fileName, dateUpload
                              sort = 'dateUploaded+desc',
                              start ="0",
                              rows = "1500"),
                         as="data.frame")
```

Resources for R

Learning

The following online books are useful for expanding your R knowledge and skills:

- the most recent ADC training materials
  - The cleaning and data manipulation section is useful for working with attribute tables
- Efficient R Programming
  - In particular Chapter 3 Efficient Programming
- R for Data Science
  - Section on Strings
- R Packages
  - contributing to `arcticdatatutils`, `datamgmt` and `EML`
- Advanced R
  - Object-oriented programming in R for S4 to understand how `datapack` and `dataone` packages are written
- bookdown: Authoring Books and Technical Documents with R Markdown
- formatting, troubleshooting and updating the training document

Others

- Hands-On Programming with R
- R Programming for Data Science
- Exploratory Data Analysis with R
- Mastering Software Development in R
- Geocomputation with R
- R Markdown: The Definitive Guide
- The Tidyverse Style Guide

The RStudio cheatsheets are also useful references for functions in tidyverse and other packages.

Packages

The data team uses and develops a number of R packages. Here is a listing and description of the main packages:

- dataone
  - reading and writing data at DataONE member nodes
  - http://doi.org/10.5063/F1M61H5X
- datapack
  - creating and managing data packages
  - https://github.com/ropensci/datapack
- EML
  - creating and editing EML metadata documents
  - https://ropensci.github.io/EML
- arcticdatautils
  - utility functions for processing data for the Arctic Data Center
  - https://nceas.github.io/arcticdatautils/
- datamgmt
  - data management utilities for curating, documenting, and publishing data (sandbox package)
  - https://nceas.github.io/datamgmt/
- metadig
  - authoring MetaDIG quality checks
  - https://github.com/NCEAS/metadig-r
- metajam
  - downloading and reading data and metadata from DataONE member nodes
  - https://nceas.github.io/metajam/