JuliaEO Workshop 2023:

Creating Packages to Retrieve/Analyze Geophysical Data

Presented By: Nathanael Wong

in collaboration with *Alexander Barth*

material at github.com/natgeo-wong/JuliaEO2023

Goal

- Introduce to you a few packages I create and use for Climate work
- But, more importantly, I want to guide people through my thought process for the ***creation*** of these packages:
  - What I have learned for the past 3 years in using Julia
  - Best-performance tips (especially when using NCDatasets.jl)
- Lots of people here are new to Julia, so I thought I’d give a bit more of a behind-the-scenes look into the creation of packages

Outline

- Managing package and project environments
  - Standardizing projects across platforms using DrWatson.jl

- An Introduction to the following packages:
  - **GeoRegions.jl**
  - **NASAPrecipitation.jl**
  - ERA5Reanalysis.jl

- Using GeoRegions.jl as a steppingstone to create packages:
  - For this workshop: sketch an outline for MERRA2Reanalysis.jl

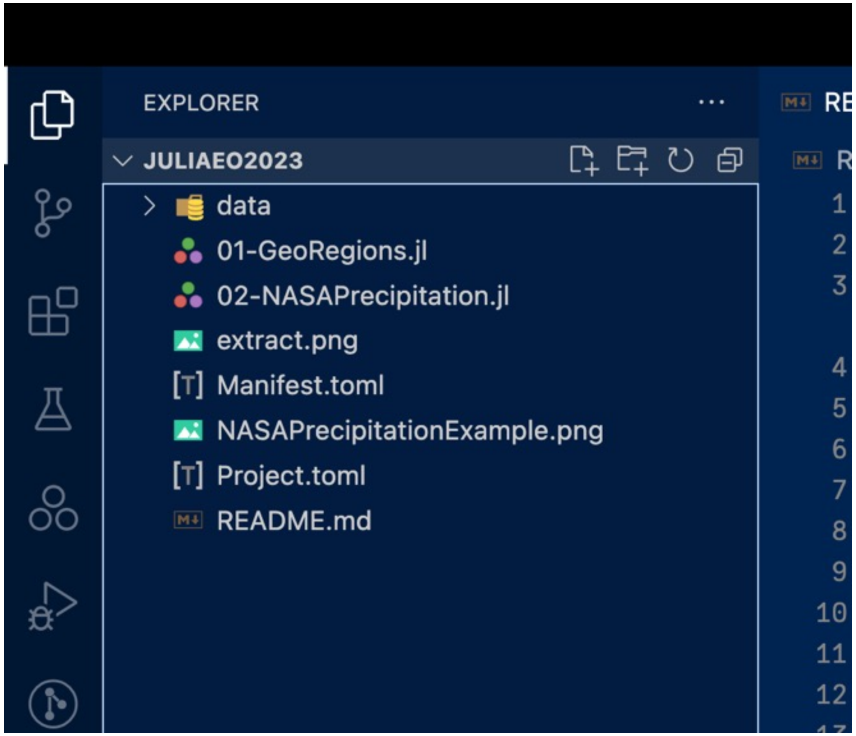
An Introduction to DrWatson.jl

STANDARDIZING PROJECTS ACROSS PLATFORMS

Managing Projects with DrWatson.jl

- A Julia project / environment defines and controls
  - Packages used in the project, and version bounds
  - Exact status of package (#main, version, etc.)

- Defined using both Project.toml and Manifest.toml
  - Project.toml contains package list and is ***always*** necessary
  - Manifest.toml contains package ***<u>and dependency</u>*** information



EXPLORER

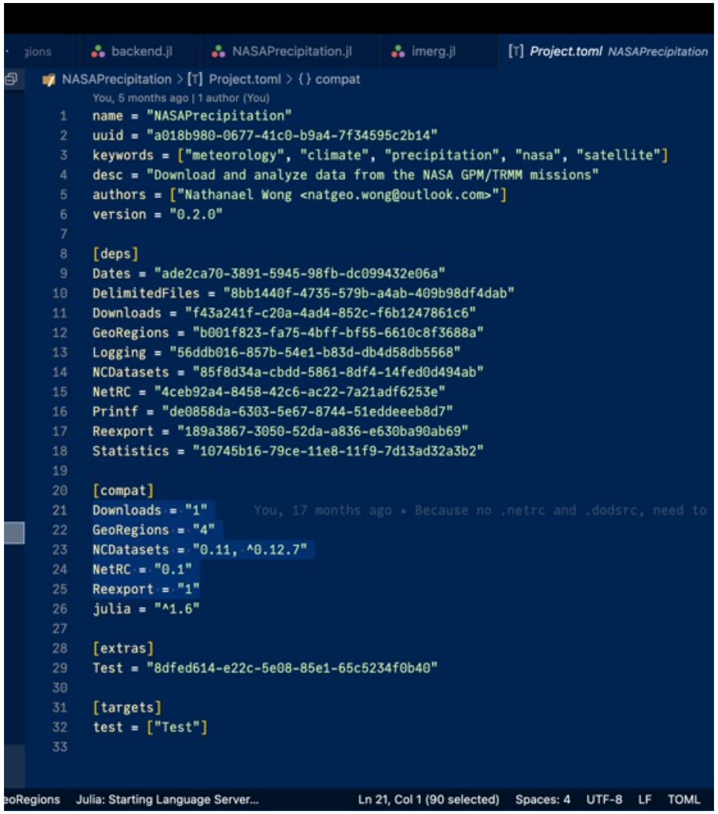
JULIAEO2023

- data
- 01-GeoRegions.jl
- 02-NASAPrecipitation.jl
- extract.png
- Manifest.toml
- NASAPrecipitationExample.png
- Project.toml
- README.md

Managing Projects with DrWatson.jl

- A Julia project / environment defines and controls
  - Packages used in the project, and version bounds
  - Exact status of package (#main, version, etc.)

- Defined using both Project.toml and Manifest.toml
  - Project.toml contains package list and is ***always*** necessary
  - Manifest.toml contains package ***and dependency*** information

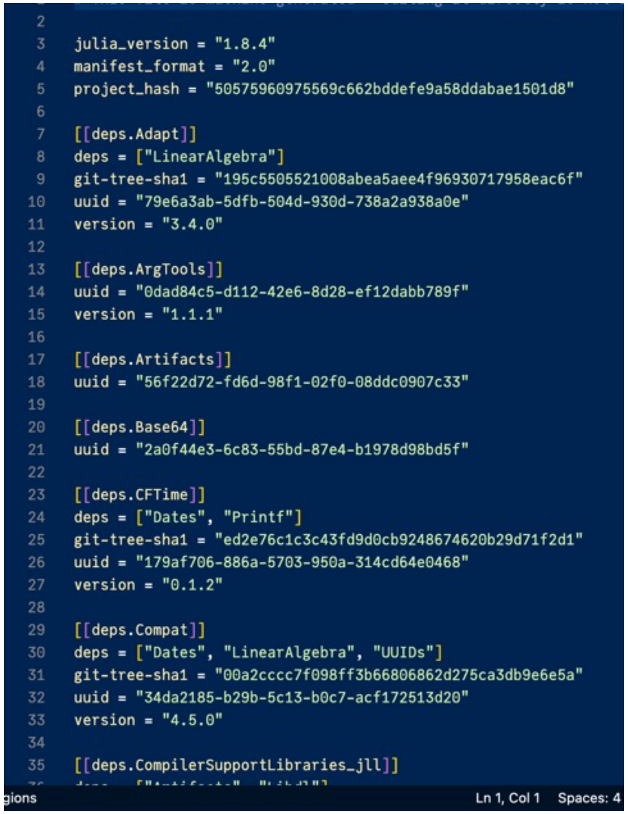

gions
backend.jl
NASAPrecipitation.jl
imerg.jl
[T] Project.toml NASAPrecipitation
NASAPrecipitation > [T] Project.toml > { } compat
You, 5 months ago | 1 author (You)
You, 17 months ago • Because no .netrc and .dodsrc, need to
eoRegions
Julia: Starting Language Server...
Ln 21, Col 1 (90 selected)
Spaces: 4
UTF-8
LF
TOML

```toml
name = "NASAPrecipitation"
uuid = "a018b980-0677-41c0-b9a4-7f34595c2b14"
keywords = ["meteorology", "climate", "precipitation", "nasa", "satellite"]
desc = "Download and analyze data from the NASA GPM/TRMM missions"
authors = ["Nathanael Wong <natgeo.wong@outlook.com>"]
version = "0.2.0"

[deps]
Dates = "ade2ca70-3891-5945-98fb-dc099432e06a"
DelimitedFiles = "8bb1440f-4735-579b-a4ab-409b98df4dab"
Downloads = "f43a241f-c20a-4ad4-852c-f6b1247861c6"
GeoRegions = "b001f823-fa75-4bff-bf55-6610c8f3688a"
Logging = "56ddb016-857b-54e1-b83d-db4d58db5568"
NCDatasets = "85f8d34a-cbdd-5861-8df4-14fed0d494ab"
NetRC = "4ceb92a4-8458-42c6-ac22-7a21adf6253e"
Printf = "de0858da-6303-5e67-8744-51eddeeeb8d7"
Reexport = "189a3867-3050-52da-a836-e630ba90ab69"
Statistics = "10745b16-79ce-11e8-11f9-7d13ad32a3b2"

[compat]
Downloads = "1"
GeoRegions = "4"
NCDatasets = "0.11, ^0.12.7"
NetRC = "0.1"
Reexport = "1"
julia = "^1.6"

[extras]
Test = "8dfed614-e22c-5e08-85e1-65c5234f0b40"

[targets]
test = ["Test"]
```

Managing Projects with DrWatson.jl

- A Julia project / environment defines and controls
  - Packages used in the project, and version bounds
  - Exact status of package (#main, version, etc.)

- Defined using both Project.toml and Manifest.toml
  - Project.toml contains package list and is ***always*** necessary
  - Manifest.toml contains package ***and dependency*** information



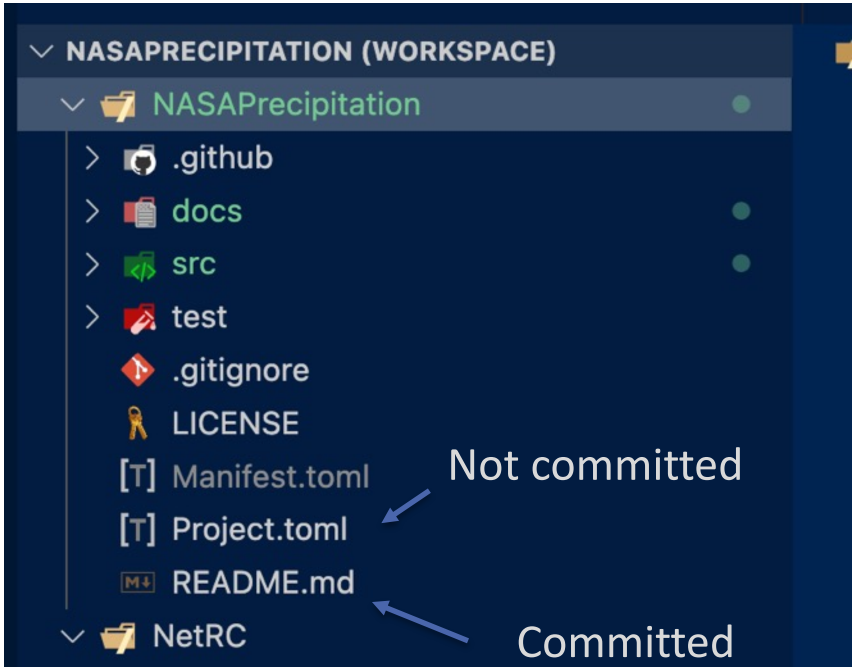
```toml

julia_version = "1.8.4"
manifest_format = "2.0"
project_hash = "50575960975569c662bddefe9a58ddabae1501d8"

[[deps.Adapt]]
deps = ["LinearAlgebra"]
git-tree-sha1 = "195c5505521008abea5aee4f96930717958eac6f"
uuid = "79e6a3ab-5dfb-504d-930d-738a2a938a0e"
version = "3.4.0"

[[deps.ArgTools]]
uuid = "0dad84c5-d112-42e6-8d28-ef12dabb789f"
version = "1.1.1"

[[deps.Artifacts]]
uuid = "56f22d72-fd6d-98f1-02f0-08ddc0907c33"

[[deps.Base64]]
uuid = "2a0f44e3-6c83-55bd-87e4-b1978d98bd5f"

[[deps.CFTime]]
deps = ["Dates", "Printf"]
git-tree-sha1 = "ed2e76c1c3c43fd9d0cb9248674620b29d71f2d1"
uuid = "179af706-886a-5703-950a-314cd64e0468"
version = "0.1.2"

[[deps.Compat]]
deps = ["Dates", "LinearAlgebra", "UUIDs"]
git-tree-sha1 = "00a2cccc7f098ff3b66806862d275ca3db9e6e5a"
uuid = "34da2185-b29b-5c13-b0c7-acf172513d20"
version = "4.5.0"

[[deps.CompilerSupportLibraries_jll]]
```

Managing Projects with DrWatson.jl

- For exact package/project duplicates, you need a ***consistent Manifest.toml*** across different platforms

- You will notice when creating packages, that ***only the Project.toml*** is committed into the GitHub, not the Manifest.toml

- But, in the repository you have cloned for this session, the ***Manifest.toml*** is also committed.



Not committed

Committed

Managing Projects with DrWatson.jl

- Use DrWatson.jl (created by George Datseris) to create and manage project workflows
  - https://juliadynamics.github.io/DrWatson.jl/stable/
  - You may want to adjust the .git and .gitignore depending on your requirements

- By default, DrWatson.jl commits the Manifest.toml
  - I generally do not feel the need to until I have a ***final*** project version
  - But for reproducibility purposes (i.e. if you are publishing this project), you should commit Manifest.toml

- (If you haven't realized by now, familiarity with Git is crucial)

Managing Projects with DrWatson.jl

- Exercise time!

- Use DrWatson.jl to create a package called “TestProject”
  - Hint: Use the function ***initialize_project***

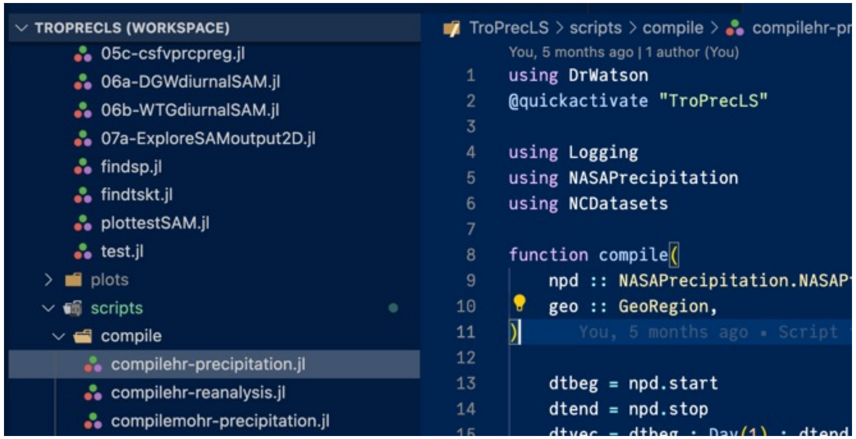
- In the default environment, install NASAPrecipitation v0.1
- In the “TestProject” environment, install NASAPrecipitation v0.2

- Compare and contrast the two (use two different Julia windows)
  - How do you call the different datasets? What is the difference in keywords?

Managing Projects with DrWatson.jl

- To run a script in a ***project*** environment (not the base environment), use the function ***@quickactivate***

- This will activate the Project.toml for the project
  - Loads the packages for the ***project*** and not the main environment



TROPRECLS (WORKSPACE)
05c-csfvprcpreg.jl
06a-DGWdiurnalSAM.jl
06b-WTGdiurnalSAM.jl
07a-ExploreSAMoutput2D.jl
findsp.jl
findtskt.jl
plottestSAM.jl
test.jl
plots
scripts
compile
compilehr-precipitation.jl
compilehr-reanalysis.jl
compilemohr-precipitation.jl

TroPrecLS > scripts > compile > compilehr-pr

You, 5 months ago | 1 author (You)

```julia
using DrWatson
@quickactivate "TroPrecLS"

using Logging
using NASAPrecipitation
using NCDatasets

function compile(
    npd :: NASAPrecipitation.NASAP
    geo :: GeoRegion,
)        You, 5 months ago • Script

    dtbeg = npd.start
    dtend = npd.stop
    dtvec = dtbeg : Day(1) : dtend
```

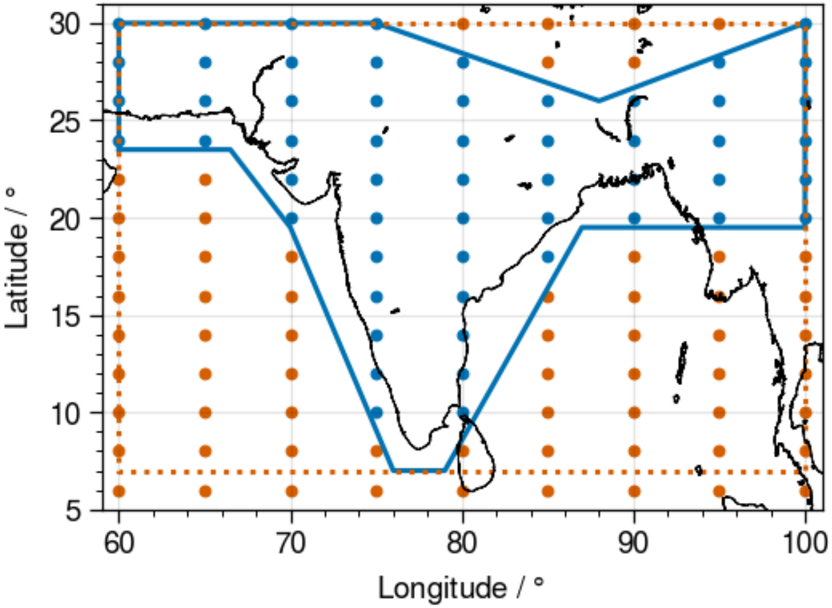
To GeoRegions.jl and NASAPrecipitation.jl

PACKAGE INTRODUCTION

GeoRegions.jl

- Deals with ***gridded*** data (preferably ***rectilinear grids***)

- Specify a Geographic Area:
  - ID
  - Name
  - Parent Region (default is GLB)
  - [N,S,E,W] coordinates ***or*** longitude/latitude vectors specifying a **shape**



GeoRegions.jl

- Open the notebook 01-georegions.jl and test out the functionality
  - Defining GeoRegions
  - Extracting data for a given GeoRegion

- This allows me to define and download regional data from online datasets (e.g. ERA5, GPM, MERRA2)
  - Use the GeoRegion ID to specify the region

- You can also expand upon it (e.g., ERA5Region type in ERA5Reanalysis.jl, which includes resolution information)

GeoRegions.jl

- Let us dive into the package itself

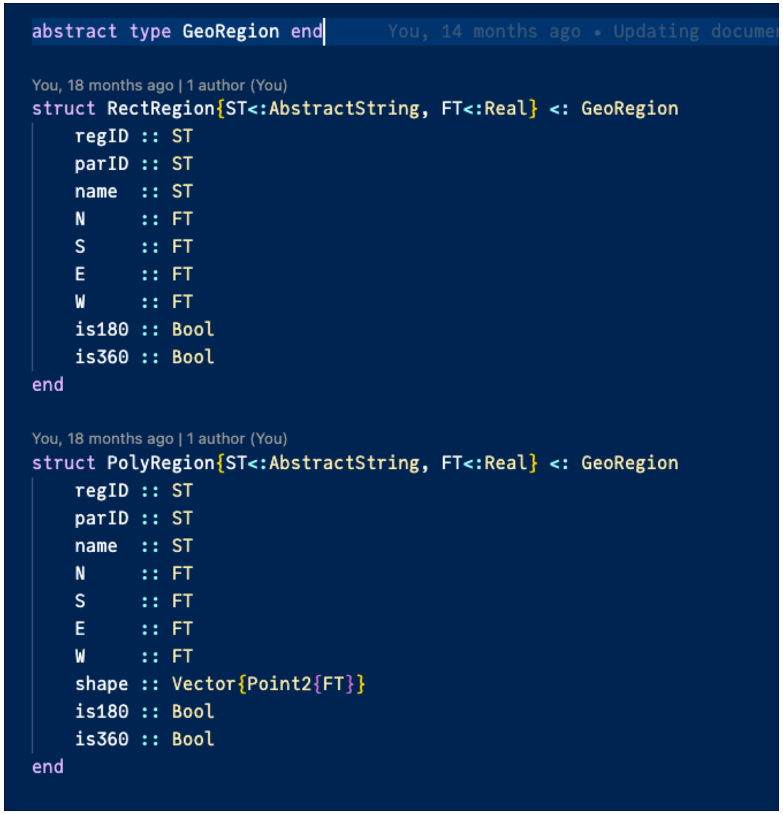
- Go into the cloned project folder and type
  -]dev GeoRegions
  - You should have a local copy of GeoRegions.jl in ~/.julia/dev/GeoRegions

- The `GeoRegion` abstract type is defined in src/GeoRegion.jl

GeoRegions.jl

- We see that the information for the GeoRegions is contained in **types**
  - Important in determining functions that can be used
  - Similar to object-orienting?

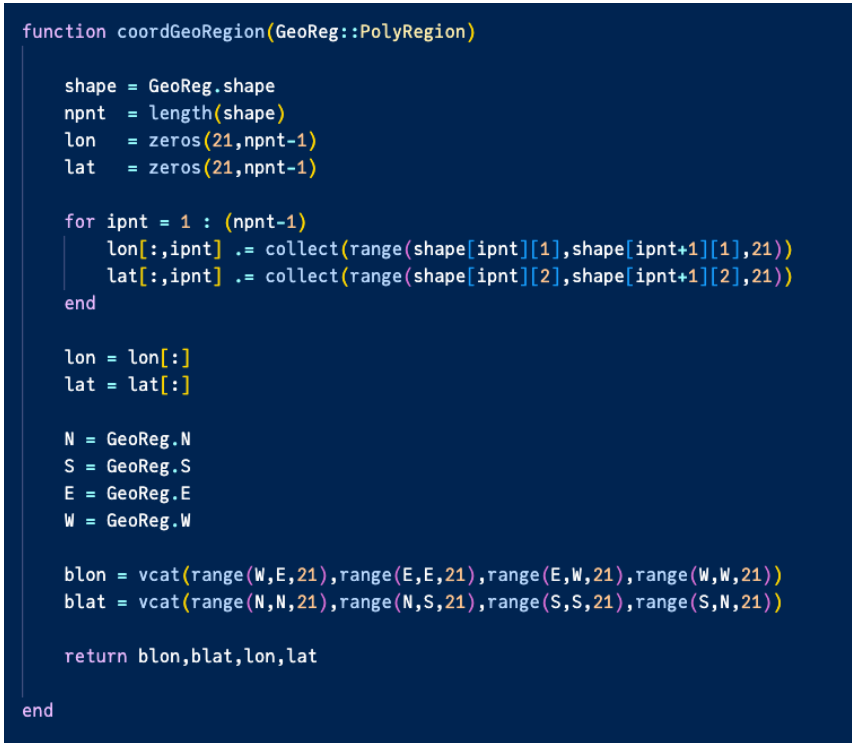
- Defining **types** is key in many of my packages
  - For example, in NASAPrecipitation.jl each dataset has its own type



```
abstract type GeoRegion end        You, 14 months ago • Updating documen

You, 18 months ago | 1 author (You)
struct RectRegion{ST<:AbstractString, FT<:Real} <: GeoRegion
    regID :: ST
    parID :: ST
    name  :: ST
    N     :: FT
    S     :: FT
    E     :: FT
    W     :: FT
    is180 :: Bool
    is360 :: Bool
end

You, 18 months ago | 1 author (You)
struct PolyRegion{ST<:AbstractString, FT<:Real} <: GeoRegion
    regID :: ST
    parID :: ST
    name  :: ST
    N     :: FT
    S     :: FT
    E     :: FT
    W     :: FT
    shape :: Vector{Point2{FT}}
    is180 :: Bool
    is360 :: Bool
end
```

GeoRegions.jl

- For example, the same function can result in different output for different ***types*** of GeoRegions



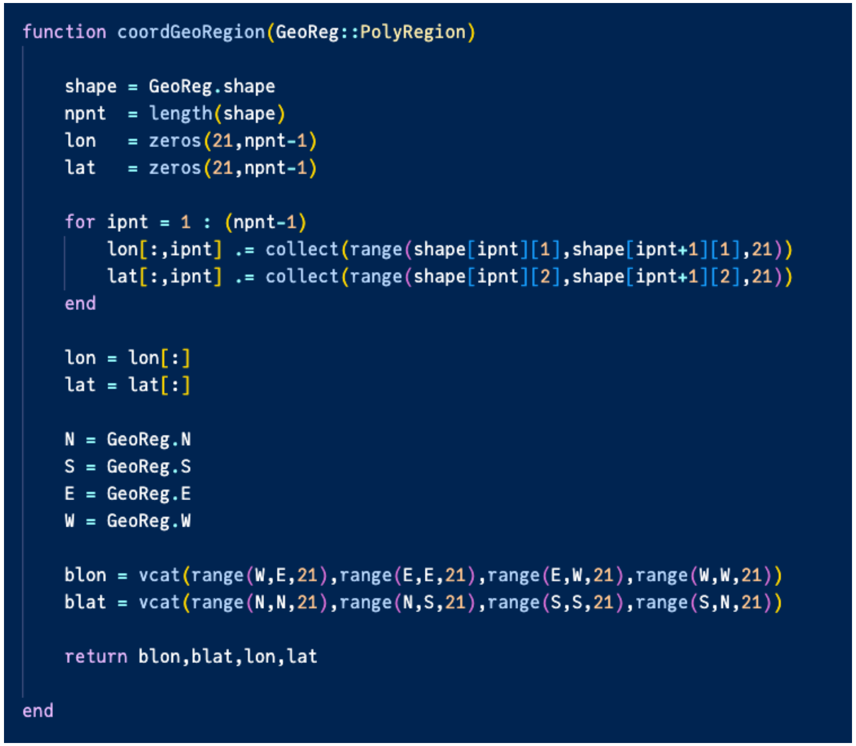
```julia
function coordGeoRegion(GeoReg::PolyRegion)

    shape = GeoReg.shape
    npnt  = length(shape)
    lon   = zeros(21,npnt-1)
    lat   = zeros(21,npnt-1)

    for ipnt = 1 : (npnt-1)
        lon[:,ipnt] .= collect(range(shape[ipnt][1],shape[ipnt+1][1],21))
        lat[:,ipnt] .= collect(range(shape[ipnt][2],shape[ipnt+1][2],21))
    end

    lon = lon[:]
    lat = lat[:]

    N = GeoReg.N
    S = GeoReg.S
    E = GeoReg.E
    W = GeoReg.W

    blon = vcat(range(W,E,21),range(E,E,21),range(E,W,21),range(W,W,21))
    blat = vcat(range(N,N,21),range(N,S,21),range(S,S,21),range(S,N,21))

    return blon,blat,lon,lat

end
```

GeoRegions.jl

- For example, the same function can result in different output for different ***types*** of GeoRegions

- Example, coordGeoRegion returns different outputs for different types of GeoRegions

- Same concept when you want to download different datasets from the same repository



```julia
function coordGeoRegion(GeoReg::PolyRegion)

    shape = GeoReg.shape
    npnt  = length(shape)
    lon   = zeros(21,npnt-1)
    lat   = zeros(21,npnt-1)

    for ipnt = 1 : (npnt-1)
        lon[:,ipnt] .= collect(range(shape[ipnt][1],shape[ipnt+1][1],21))
        lat[:,ipnt] .= collect(range(shape[ipnt][2],shape[ipnt+1][2],21))
    end

    lon = lon[:]
    lat = lat[:]

    N = GeoReg.N
    S = GeoReg.S
    E = GeoReg.E
    W = GeoReg.W

    blon = vcat(range(W,E,21),range(E,E,21),range(E,W,21),range(W,W,21))
    blat = vcat(range(N,N,21),range(N,S,21),range(S,S,21),range(S,N,21))

    return blon,blat,lon,lat

end
```

GeoRegions.jl

- You can also download ETOPO topography masks using GeoRegions.jl using getLandSea
  - Functionality can be expanded to other land-sea masks for different datasets (e.g. IMERG Land-Sea masks)

- An example of **extending** a function using new packages
  - The getLandSea functionality is extended by the NASAPrecipitation.jl and ERA5Reanalysis.jl packages

NASAPrecipitation.jl

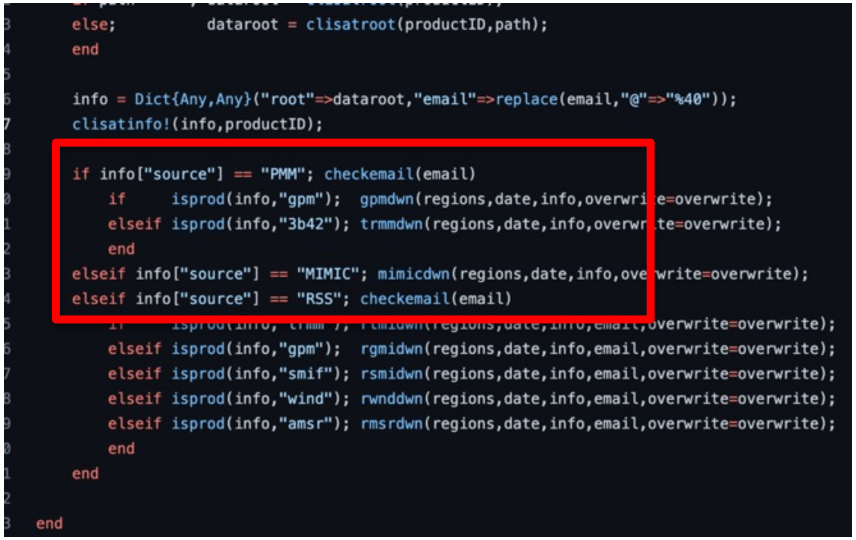
- Accesses and downloads GPM and TRMM datasets from NASA OPeNDAP servers (requires earthdata login)
  - GPM IMERG Half-Hourly, Daily and Monthly
  - TRMM 3-Hourly, Daily and Monthly

- Specify the following:
  - Start date
  - Stop date
  - Directory

- Open the notebook 02-NASAPrecipitation.jl

NASAPrecipitation.jl

- Many different datasets (Early, Late, Final) over different timescales
  - Each is defined by a ***type*** that contains critical information
  - Easier than using ***dicts***

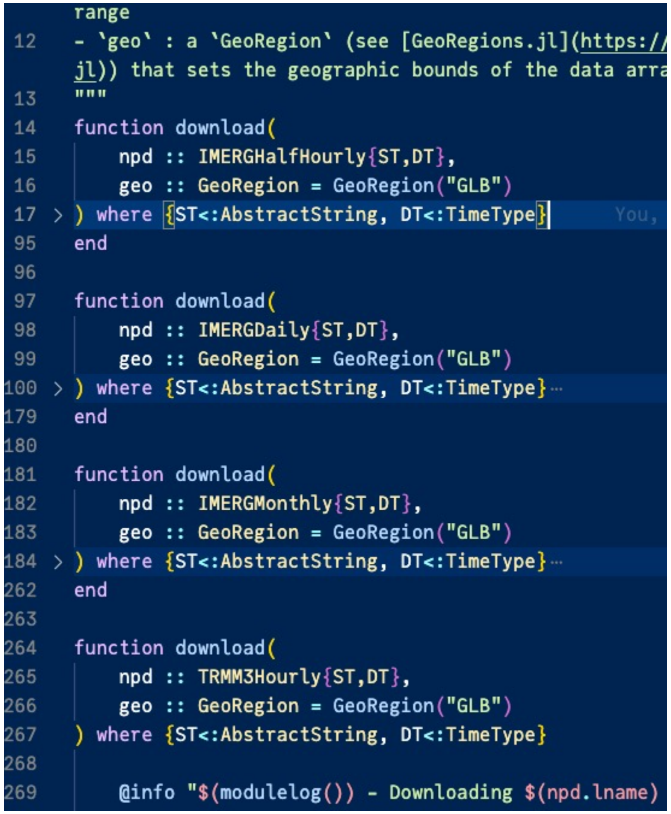


```
    else;          dataroot = clisatroot(productID,path);
    end

    info = Dict{Any,Any}("root"=>dataroot,"email"=>replace(email,"@"=>"%40"));
    clisatinfo!(info,productID);

    if info["source"] == "PMM"; checkemail(email)
        if     isprod(info,"gpm");  gpmdwn(regions,date,info,overwrite=overwrite);
        elseif isprod(info,"3b42"); trmmdwn(regions,date,info,overwrite=overwrite);
        end
    elseif info["source"] == "MIMIC"; mimicdwn(regions,date,info,overwrite=overwrite);
    elseif info["source"] == "RSS"; checkemail(email)
        if     isprod(info,"trmm"); rtmidwn(regions,date,info,email,overwrite=overwrite);
        elseif isprod(info,"gpm");  rgmidwn(regions,date,info,email,overwrite=overwrite);
        elseif isprod(info,"smif"); rsmidwn(regions,date,info,email,overwrite=overwrite);
        elseif isprod(info,"wind"); rwnddwn(regions,date,info,email,overwrite=overwrite);
        elseif isprod(info,"amsr"); rmsrdwn(regions,date,info,email,overwrite=overwrite);
        end
    end

end
```

NASAPrecipitation.jl

- Many different datasets (Early, Late, Final) over different timescales
  - Each is defined by a ***type*** that contains critical information
  - Easier than using ***dicts***

- Contrast NASAPrecipitation.jl to ClimateSatellite.jl (the predecessor)
  - Extending function methods

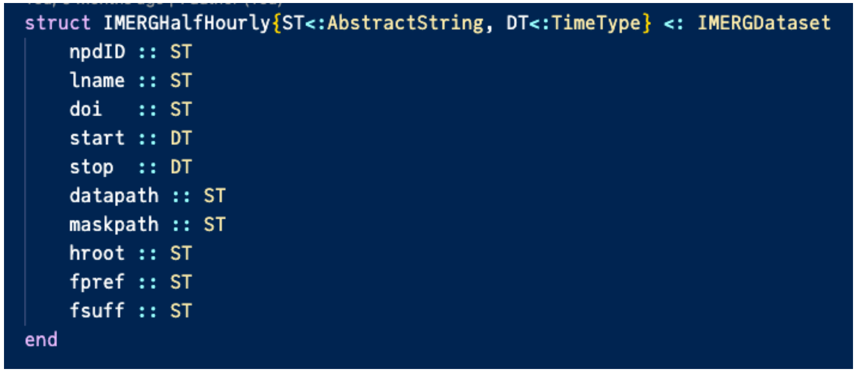


```julia
range
- `geo` : a `GeoRegion` (see [GeoRegions.jl](https://
jl)) that sets the geographic bounds of the data arra
"""
function download(
    npd :: IMERGHalfHourly{ST,DT},
    geo :: GeoRegion = GeoRegion("GLB")
) where {ST<:AbstractString, DT<:TimeType}        You,
end

function download(
    npd :: IMERGDaily{ST,DT},
    geo :: GeoRegion = GeoRegion("GLB")
) where {ST<:AbstractString, DT<:TimeType}⋯
end

function download(
    npd :: IMERGMonthly{ST,DT},
    geo :: GeoRegion = GeoRegion("GLB")
) where {ST<:AbstractString, DT<:TimeType}⋯
end

function download(
    npd :: TRMM3Hourly{ST,DT},
    geo :: GeoRegion = GeoRegion("GLB")
) where {ST<:AbstractString, DT<:TimeType}

    @info "$(modulelog()) - Downloading $(npd.lname)
```

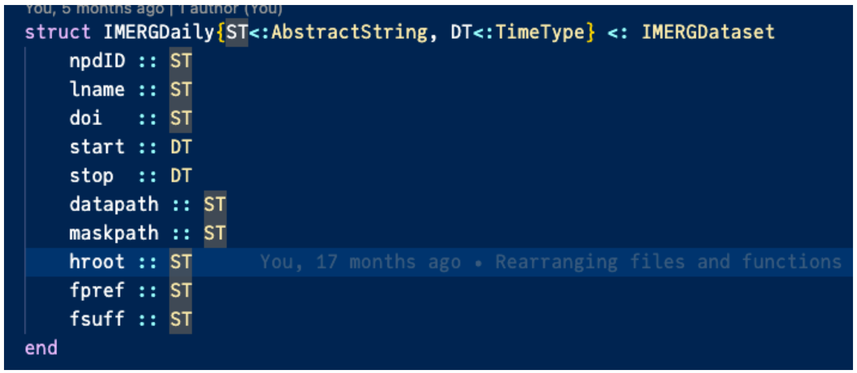
NASAPrecipitation.jl

- Many different datasets (Early, Late, Final) over different timescales
  - Each is defined by a ***type*** that contains critical information
  - Easier than using ***dicts***

- Contrast NASAPrecipitation.jl to ClimateSatellite.jl (the predecessor)
  - Extending function methods



```julia
struct IMERGHalfHourly{ST<:AbstractString, DT<:TimeType} <: IMERGDataset
    npdID :: ST
    lname :: ST
    doi   :: ST
    start :: DT
    stop  :: DT
    datapath :: ST
    maskpath :: ST
    hroot :: ST
    fpref :: ST
    fsuff :: ST
end
```

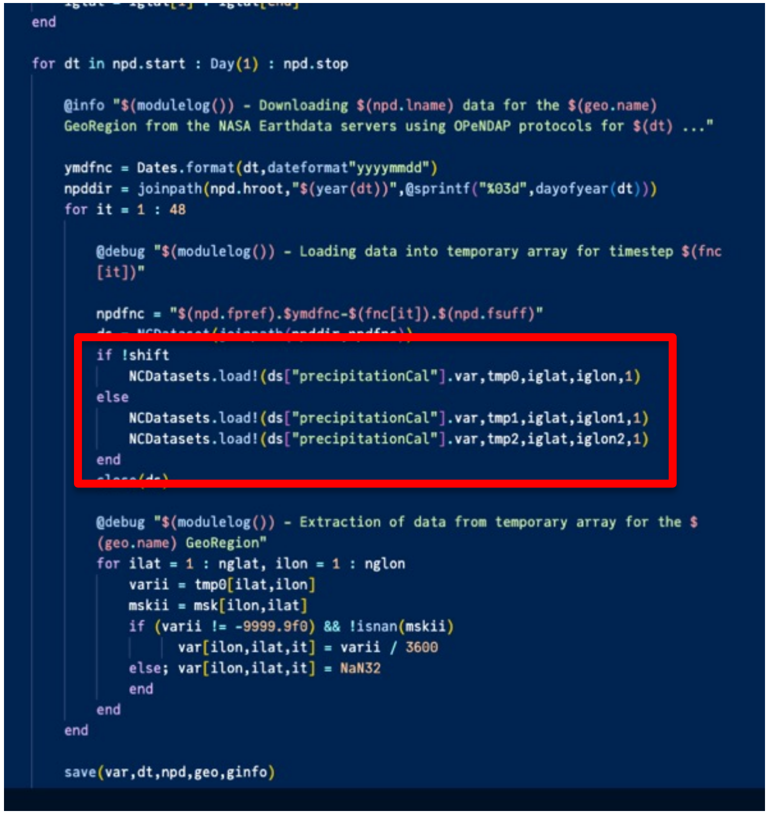


You, 5 months ago | 1 author (You)

```julia
struct IMERGDaily{ST<:AbstractString, DT<:TimeType} <: IMERGDataset
    npdID :: ST
    lname :: ST
    doi   :: ST
    start :: DT
    stop  :: DT
    datapath :: ST
    maskpath :: ST
    hroot :: ST        You, 17 months ago • Rearranging files and functions
    fpref :: ST
    fsuff :: ST
end
```

NASAPrecipitation.jl

- Utilizes the power of NCDatasets.jl
  - In-place loading of arrays
  - Allows for remote access of NetCDF dataset
- Saves memory (in-place loading)
- Saves space (no temporary files)
  - ***Note***: downloading may be slower than direct HTTPs download
- See example on right

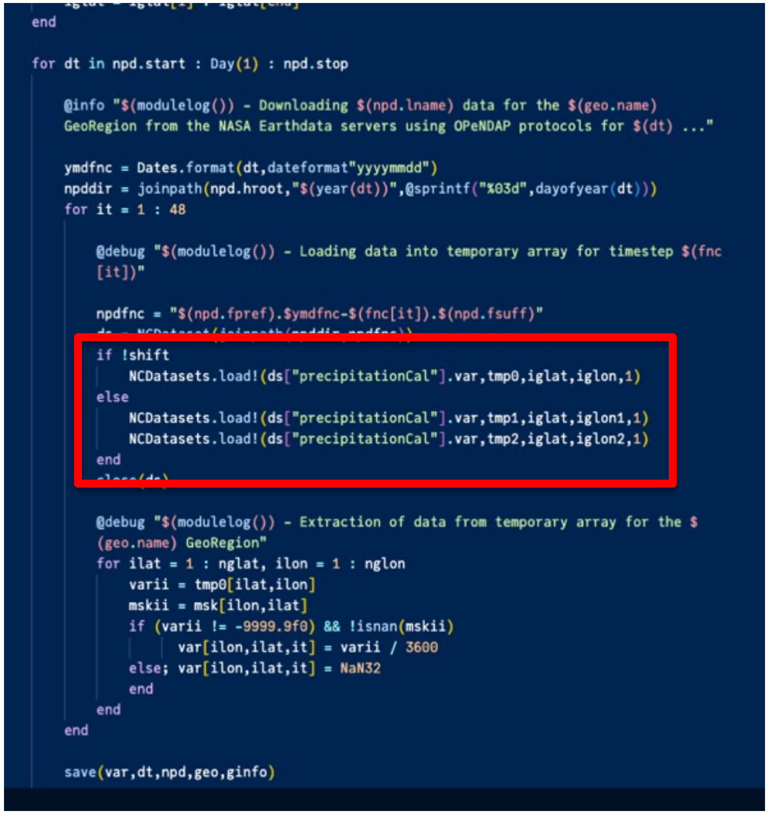


```
    iglat = iglat[1] : iglat[end]
end

for dt in npd.start : Day(1) : npd.stop

    @info "$(modulelog()) - Downloading $(npd.lname) data for the $(geo.name)
    GeoRegion from the NASA Earthdata servers using OPeNDAP protocols for $(dt) ..."

    ymdfnc = Dates.format(dt,dateformat"yyyymmdd")
    npddir = joinpath(npd.hroot,"$(year(dt))",@sprintf("%03d",dayofyear(dt)))
    for it = 1 : 48

        @debug "$(modulelog()) - Loading data into temporary array for timestep $(fnc
        [it])"

        npdfnc = "$(npd.fpref).$ymdfnc-$(fnc[it]).$(npd.fsuff)"
        ds = NCDataset(joinpath(npddir,npdfnc))
        if !shift
            NCDatasets.load!(ds["precipitationCal"].var,tmp0,iglat,iglon,1)
        else
            NCDatasets.load!(ds["precipitationCal"].var,tmp1,iglat,iglon1,1)
            NCDatasets.load!(ds["precipitationCal"].var,tmp2,iglat,iglon2,1)
        end
        close(ds)

        @debug "$(modulelog()) - Extraction of data from temporary array for the $
        (geo.name) GeoRegion"
        for ilat = 1 : nglat, ilon = 1 : nglon
            varii = tmp0[ilat,ilon]
            mskii = msk[ilon,ilat]
            if (varii != -9999.9f0) && !isnan(mskii)
                var[ilon,ilat,it] = varii / 3600
            else; var[ilon,ilat,it] = NaN32
            end
        end
    end

    save(var,dt,npd,geo,ginfo)
```

NASAPrecipitation.jl

- Tip for package creation:
  - Compiled for-loops are fast → item-by-item manipulation
  - Broadcasting uses memory

- Tip for NCDatasets.jl usage:
  - Opening large NetCDF files repeatedly may cause memory overflow
  - Computer cannot purge memory fast enough before loading next file
  - In-place loading helps prevent this



```julia
    iglat = iglat[1] : iglat[end]
end

for dt in npd.start : Day(1) : npd.stop

    @info "$(modulelog()) - Downloading $(npd.lname) data for the $(geo.name)
    GeoRegion from the NASA Earthdata servers using OPeNDAP protocols for $(dt) ..."

    ymdfnc = Dates.format(dt,dateformat"yyyymmdd")
    npddir = joinpath(npd.hroot,"$(year(dt))",@sprintf("%03d",dayofyear(dt)))
    for it = 1 : 48

        @debug "$(modulelog()) - Loading data into temporary array for timestep $(fnc
        [it])"

        npdfnc = "$(npd.fpref).$ymdfnc-$(fnc[it]).$(npd.fsuff)"
        ds = NCDataset(joinpath(npddir,npdfnc))
        if !shift
            NCDatasets.load!(ds["precipitationCal"].var,tmp0,iglat,iglon,1)
        else
            NCDatasets.load!(ds["precipitationCal"].var,tmp1,iglat,iglon1,1)
            NCDatasets.load!(ds["precipitationCal"].var,tmp2,iglat,iglon2,1)
        end
        close(ds)

        @debug "$(modulelog()) - Extraction of data from temporary array for the $
        (geo.name) GeoRegion"
        for ilat = 1 : nglat, ilon = 1 : nglon
            varii = tmp0[ilat,ilon]
            mskii = msk[ilon,ilat]
            if (varii != -9999.9f0) && !isnan(mskii)
                    var[ilon,ilat,it] = varii / 3600
            else; var[ilon,ilat,it] = NaN32
            end
        end
    end

    save(var,dt,npd,geo,ginfo)
```

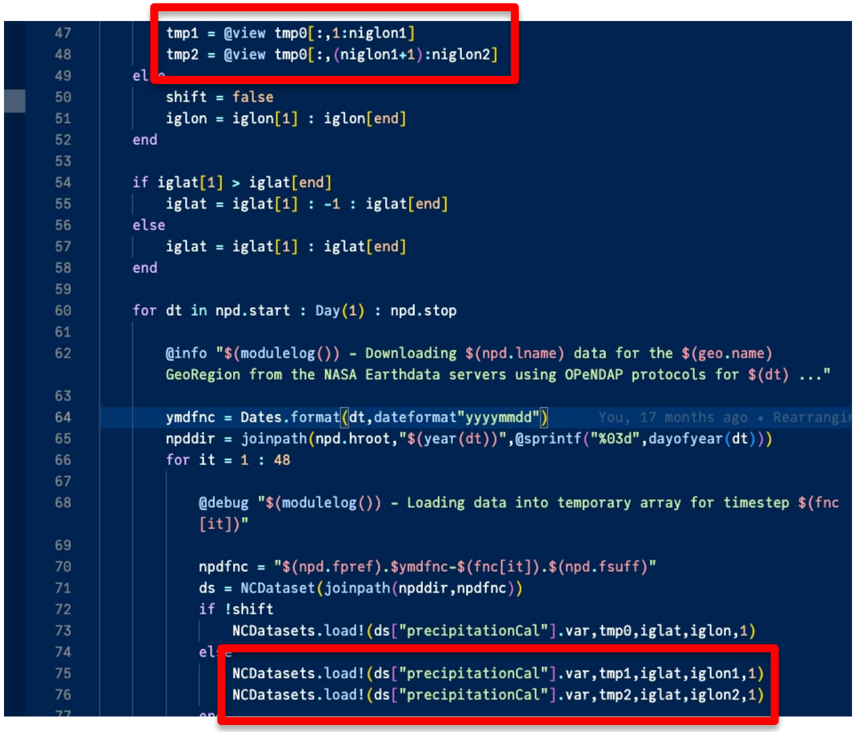
NASAPrecipitation.jl

- The power of @view
  - Allows me to split an array for memory allocation

- Caution: may need more memory to be allocated upfront

<u>but</u>

- Memory allocated is reused (so no new allocations)

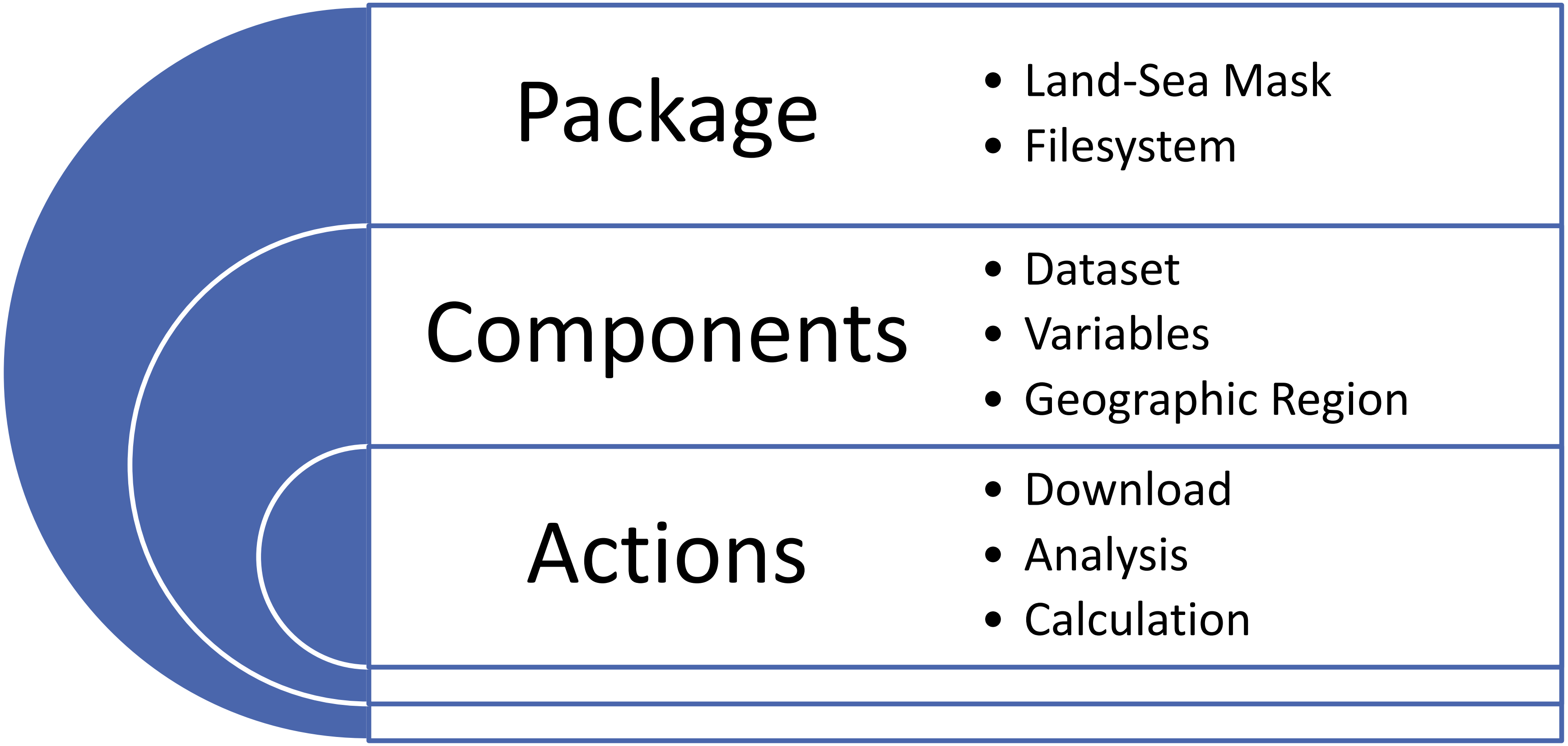


```julia
        tmp1 = @view tmp0[:,1:niglon1]
        tmp2 = @view tmp0[:,(niglon1+1):niglon2]
    el
        shift = false
        iglon = iglon[1] : iglon[end]
    end

    if iglat[1] > iglat[end]
        iglat = iglat[1] : -1 : iglat[end]
    else
        iglat = iglat[1] : iglat[end]
    end

    for dt in npd.start : Day(1) : npd.stop

        @info "$(modulelog()) - Downloading $(npd.lname) data for the $(geo.name) GeoRegion from the NASA Earthdata servers using OPeNDAP protocols for $(dt) ..."

        ymdfnc = Dates.format(dt,dateformat"yyyymmdd")        You, 17 months ago • Rearrangi
        npddir = joinpath(npd.hroot,"$(year(dt))",@sprintf("%03d",dayofyear(dt)))
        for it = 1 : 48

            @debug "$(modulelog()) - Loading data into temporary array for timestep $(fnc[it])"

            npdfnc = "$(npd.fpref).$ymdfnc-$(fnc[it]).$(npd.fsuff)"
            ds = NCDataset(joinpath(npddir,npdfnc))
            if !shift
                NCDatasets.load!(ds["precipitationCal"].var,tmp0,iglat,iglon,1)
            el
                NCDatasets.load!(ds["precipitationCal"].var,tmp1,iglat,iglon1,1)
                NCDatasets.load!(ds["precipitationCal"].var,tmp2,iglat,iglon2,1)
```

Sketching out the structure of creating a new package

CREATING A NEW PACKAGE

What comprises a package?



Package

- Land-Sea Mask
- Filesystem

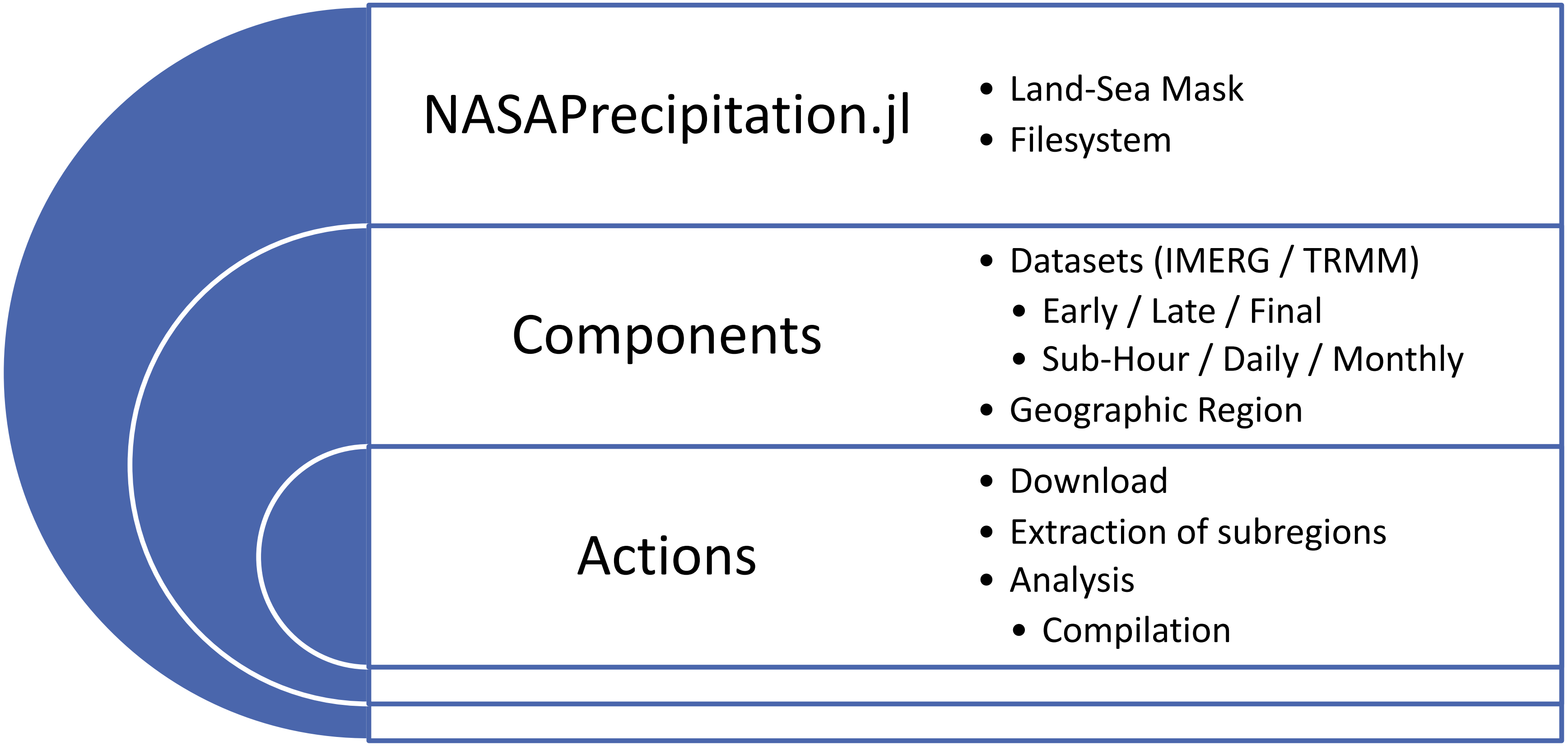
Components

- Dataset
- Variables
- Geographic Region

Actions

- Download
- Analysis
- Calculation

What comprises a package?



NASAPrecipitation.jl

- Land-Sea Mask
- Filesystem

Components

- Datasets (IMERG / TRMM)
  - Early / Late / Final
  - Sub-Hour / Daily / Monthly
- Geographic Region

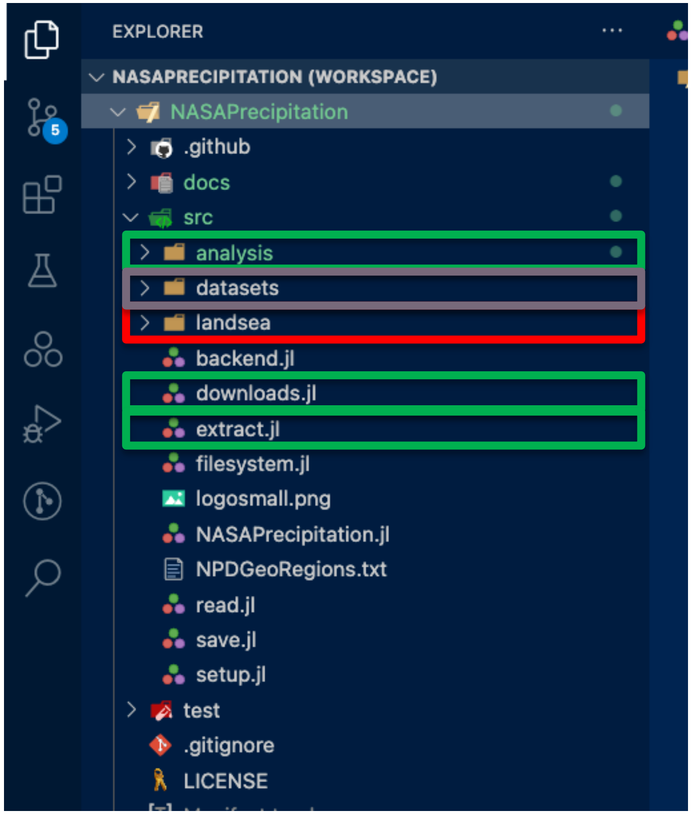
Actions

- Download
- Extraction of subregions
- Analysis
  - Compilation

What comprises a package?

- See filesystem structure on the right here
  - Red = package stuff
  - Purple = components
  - Green = actionables

- You also have miscellaneous backend items
  - Date2String functions
  - Error checks
  - Nan-means



EXPLORER

NASAPRECIPITATION (WORKSPACE)

NASAPrecipitation

.github

docs

src

analysis

datasets

landsea

backend.jl

downloads.jl

extract.jl

filesystem.jl

logosmall.png

NASAPrecipitation.jl

NPDGeoRegions.txt

read.jl

save.jl

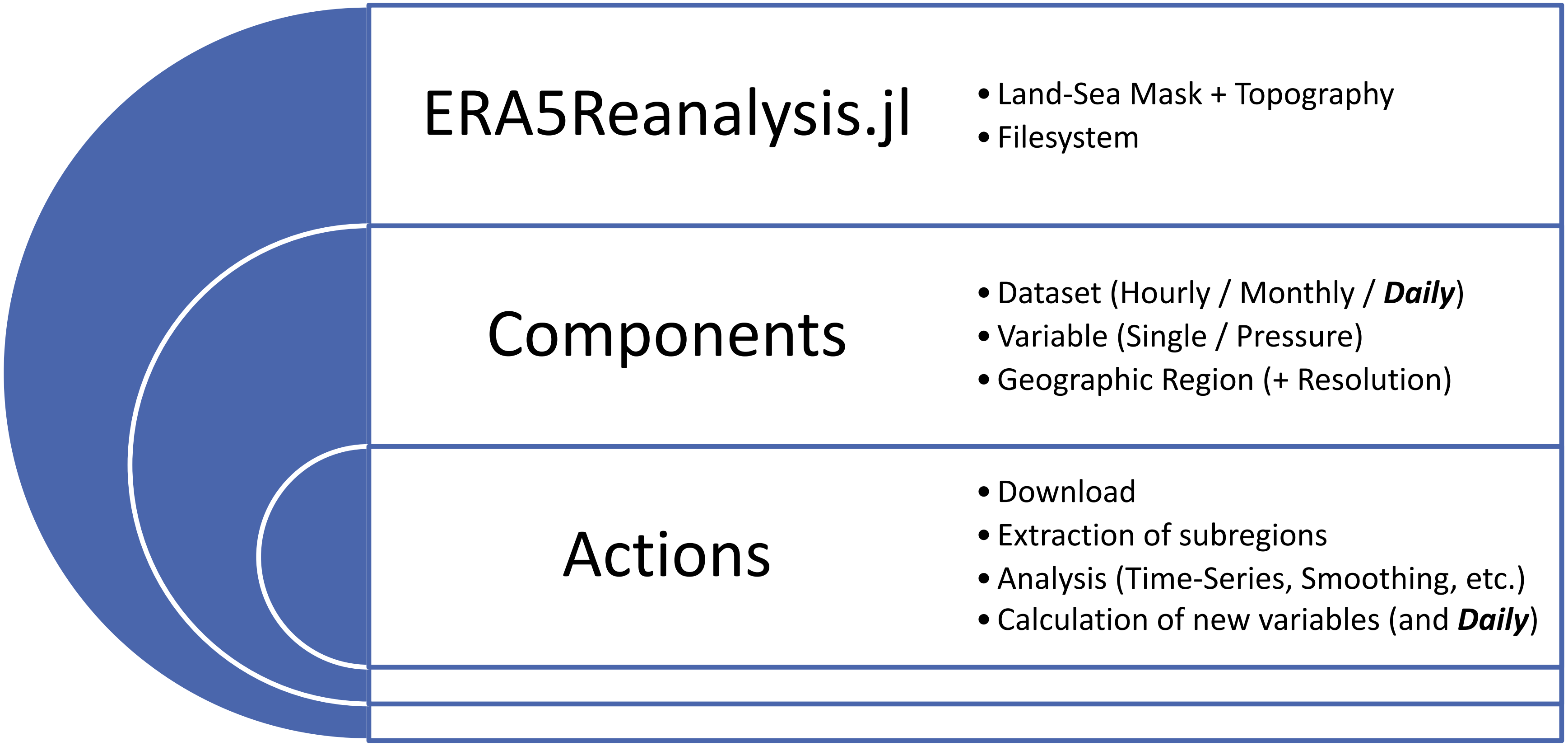
setup.jl

test

.gitignore

LICENSE

What comprises a package?



ERA5Reanalysis.jl

- Land-Sea Mask + Topography
- Filesystem

Components

- Dataset (Hourly / Monthly / ***Daily***)
- Variable (Single / Pressure)
- Geographic Region (+ Resolution)

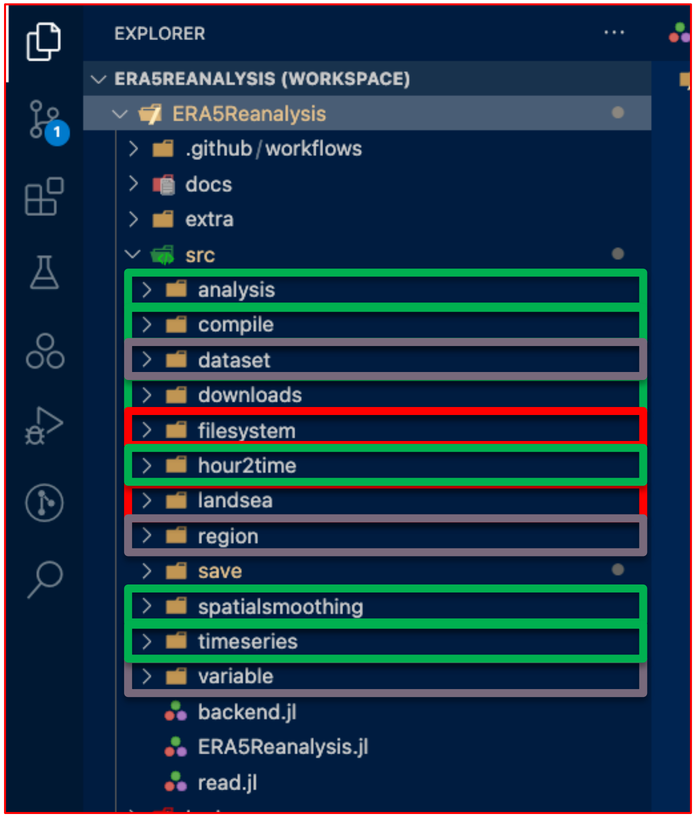
Actions

- Download
- Extraction of subregions
- Analysis (Time-Series, Smoothing, etc.)
- Calculation of new variables (and ***Daily***)

What comprises a package?

- See filesystem structure on the right here
  - Red = package stuff
  - Purple = components
  - Green = actionables

- You also have miscellaneous backend items
  - Date2String functions
  - Error checks
  - Nan-means
  - ***Real2Int*** functions



EXPLORER

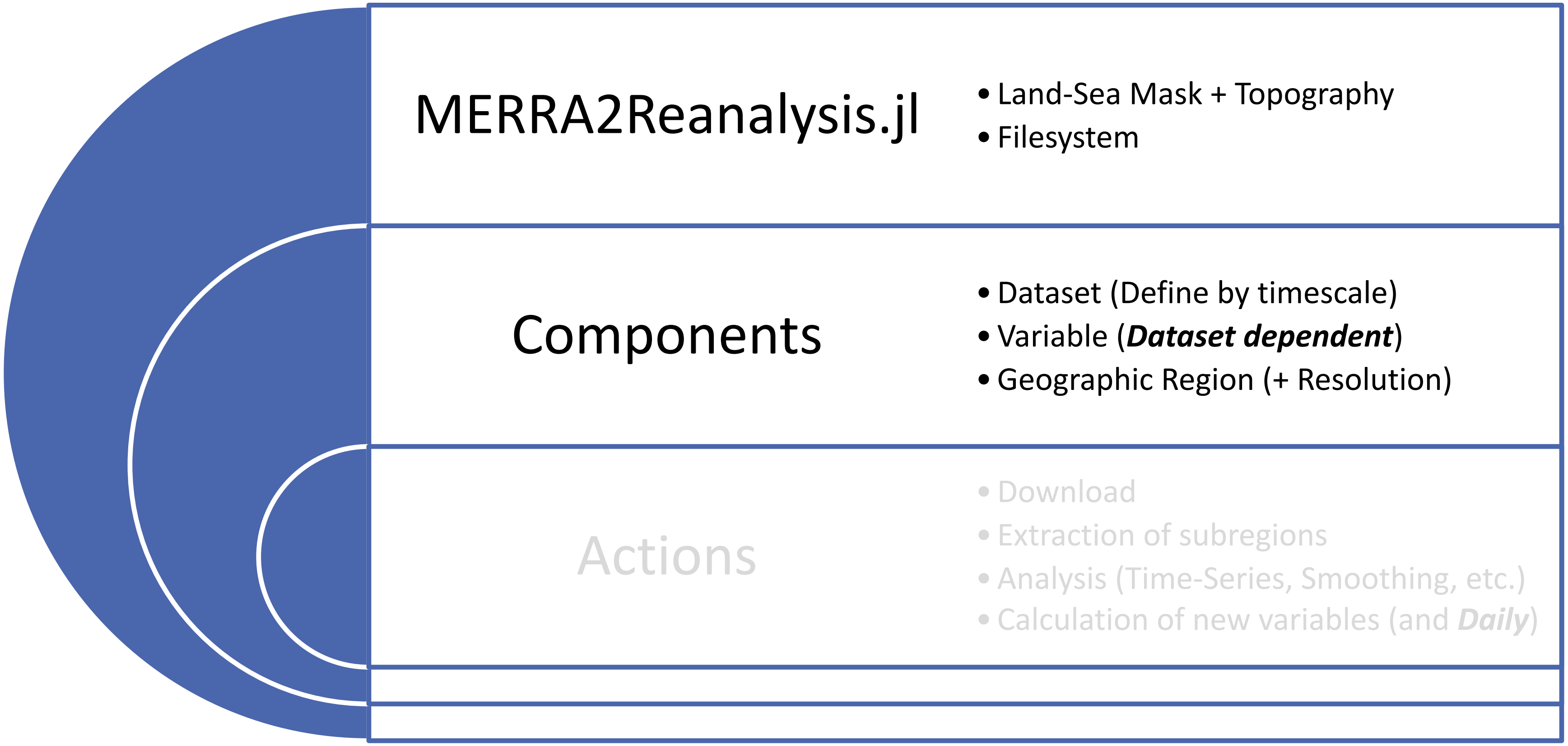
ERA5REANALYSIS (WORKSPACE)

```
ERA5Reanalysis
  .github/workflows
  docs
  extra
  src
    analysis
    compile
    dataset
    downloads
    filesystem
    hour2time
    landsea
    region
    save
    spatialsmoothing
    timeseries
    variable
    backend.jl
    ERA5Reanalysis.jl
    read.jl
```

Let’s apply this to MERRA2

- I am currently drafting this package
- Refer to the MERRA-2 file specification document:
  - https://gmao.gsfc.nasa.gov/pubs/docs/Bosilovich785.pdf
  - There are many different variables and datasets
- What are the components of MERRA-2?

What comprises a package?



MERRA2Reanalysis.jl

- Land-Sea Mask + Topography
- Filesystem

Components

- Dataset (Define by timescale)
- Variable (***Dataset dependent***)
- Geographic Region (+ Resolution)

Actions

- Download
- Extraction of subregions
- Analysis (Time-Series, Smoothing, etc.)
- Calculation of new variables (and ***Daily***)

Let’s apply this to MERRA2

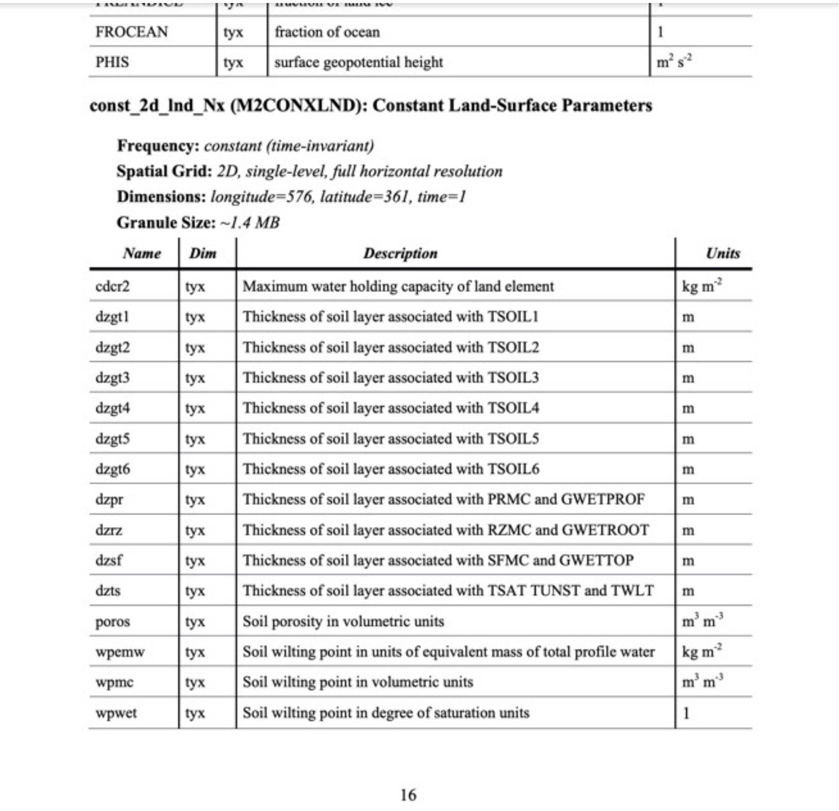
- I am currently drafting this package

- Refer to the MERRA-2 file specification document:
  - https://gmao.gsfc.nasa.gov/pubs/docs/Bosilovich785.pdf
  - There are many different variables and datasets

- What are the components of MERRA-2?
  - How do you define the datasets?
  - How do you define the variables? How are they dependent on the datasets?
    - How do you ensure that a variable is available for download in a dataset?

Let’s apply this to MERRA2

- I am currently drafting this package
- There’s no one correct answer to this question
- How do you group the different datasets?
- My current thought process: have lists of variables for each dataset?
  - Different datasets will have different variables



FRLANDICE	tyx	fraction of land ice	1
FROCEAN	tyx	fraction of ocean	1
PHIS	tyx	surface geopotential height	$m^2 s^{-2}$

const_2d_lnd_Nx (M2CONXLND): Constant Land-Surface Parameters

Frequency: *constant (time-invariant)*
Spatial Grid: *2D, single-level, full horizontal resolution*
Dimensions: *longitude=576, latitude=361, time=1*
Granule Size: *~1.4 MB*

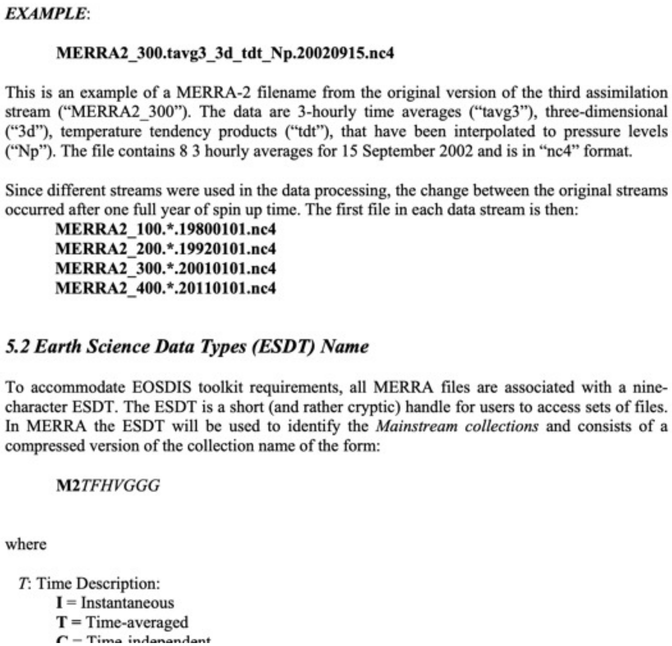
Name	*Dim*	*Description*	*Units*
cdcr2	tyx	Maximum water holding capacity of land element	kg m^{-2}
dzgt1	tyx	Thickness of soil layer associated with TSOIL1	m
dzgt2	tyx	Thickness of soil layer associated with TSOIL2	m
dzgt3	tyx	Thickness of soil layer associated with TSOIL3	m
dzgt4	tyx	Thickness of soil layer associated with TSOIL4	m
dzgt5	tyx	Thickness of soil layer associated with TSOIL5	m
dzgt6	tyx	Thickness of soil layer associated with TSOIL6	m
dzpr	tyx	Thickness of soil layer associated with PRMC and GWETPROF	m
dzrz	tyx	Thickness of soil layer associated with RZMC and GWETROOT	m
dzsf	tyx	Thickness of soil layer associated with SFMC and GWETTOP	m
dzts	tyx	Thickness of soil layer associated with TSAT TUNST and TWLT	m
poros	tyx	Soil porosity in volumetric units	$m^3 m^{-3}$
wpemw	tyx	Soil wilting point in units of equivalent mass of total profile water	kg m^{-2}
wpmc	tyx	Soil wilting point in volumetric units	$m^3 m^{-3}$
wpwet	tyx	Soil wilting point in degree of saturation units	1

16

Let’s apply this to MERRA2

- What should each dataset ***type*** contain?
  - Path
  - Date begin/end
  - DOI
  - Anything else?

- Remember, the dataset ***type*** should be able to contain information to help with the downloading



EXAMPLE:

MERRA2_300.tavg3_3d_tdt_Np.20020915.nc4

This is an example of a MERRA-2 filename from the original version of the third assimilation stream (“MERRA2_300”). The data are 3-hourly time averages (“tavg3”), three-dimensional (“3d”), temperature tendency products (“tdt”), that have been interpolated to pressure levels (“Np”). The file contains 8 3 hourly averages for 15 September 2002 and is in “nc4” format.

Since different streams were used in the data processing, the change between the original streams occurred after one full year of spin up time. The first file in each data stream is then:
MERRA2_100.*.19800101.nc4
MERRA2_200.*.19920101.nc4
MERRA2_300.*.20010101.nc4
MERRA2_400.*.20110101.nc4

5.2 Earth Science Data Types (ESDT) Name

To accommodate EOSDIS toolkit requirements, all MERRA files are associated with a nine-character ESDT. The ESDT is a short (and rather cryptic) handle for users to access sets of files. In MERRA the ESDT will be used to identify the *Mainstream collections* and consists of a compressed version of the collection name of the form:

M2*TFHVGGG*

where

T: Time Description:
I = Instantaneous
T = Time-averaged
C = Time independent

Thank You!

Any Questions?