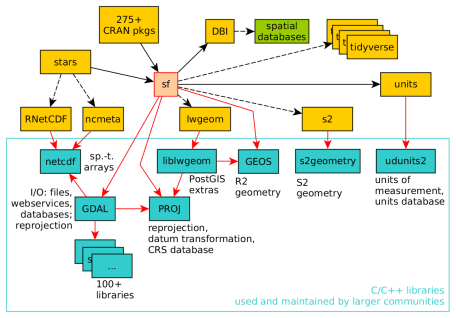
**GDAL and PROJ**

GDAL and PROJ   
are two libraries that form the basis, if not foundations, for most open  
source geospatial software, including most R packages (sf, sp, rgdal,  
and all their dependencies). The dependency for package sf is for  
instance pictured here:



Briefly:

* PROJ provides methods for coordinate representation, conversion  
  (projection) and transformation, and
* GDAL allows reading and writing of spatial raster and vector data in  
  a standardised form, and provides a high-level interface to PROJ for  
  these data structures, including the representation of coordinate  
  reference systems (CRS)

**gdalbarn**

Motivated by the need for higher precision handling of coordinate  
transformations and the wish to support for a better description of  
coordinate reference systems  
(WKT2), a  
succesful fundraising helped the implementation  
of a large number of changes in GDAL and PROJ, most notably:

* PROJ changes from (mostly) a projection library into a full geodetic  
  library, taking care of different representations of the shape of  
  the Earth (datums)
* PROJ now has the ability to choose between different transformation  
  paths (pipelines), and can report the precision obtained by each
* rather than distributing datum transformation grids to local users,  
  PROJ (7.0.0 and higher) offers access to an on-line distribution  
  network (CDN) of free transformation grids,  
  thereby allowing for local caching of portions of grids
* PROJ respects authorities (such as EPSG) for determining whether  
  coordinate pairs refer to longitude-latitude (such as 3857), or  
  latitude-longitude (such as 4326)
* GDAL offers the ability to handle coordinate pairs  
  authority-compliant (lat-long for 4326), or “traditional”  
  GIS-compliant (long-lat for 4326)
* use of so-called PROJ4-strings (like +proj=longlat +datum=WGS84)  
  are discouraged, they no longer offer sufficient description of  
  coordinate reference systems; use of +init=epsg:XXXX leads to  
  warnings
* PROJ offers access to a large number of vertical reference systems  
  and reference systems of authorities different from EPSG

**crs objects in sf**

Pre-0.9 versions of sf used crs (coordinate reference system)  
objects represented as lists with two components, epsg (possibly set  
as NA) and proj4string:

library(sf)

# Linking to GEOS 3.8.0, GDAL 3.0.2, PROJ 6.2.1

st\_crs(4326)

# Coordinate Reference System:

# EPSG: 4326

# proj4string: "+proj=longlat +datum=WGS84 +no\_defs"

now, with sf >= 0.9, crs objects are lists with two components,  
input and wkt:

library(sf)

## Linking to GEOS 3.8.0, GDAL 3.0.2, PROJ 6.2.1

(x = st\_crs(4326))

## Coordinate Reference System:

## User input: EPSG:4326

## wkt:

## GEOGCRS["WGS 84",

## DATUM["World Geodetic System 1984",

## ELLIPSOID["WGS 84",6378137,298.257223563,

## LENGTHUNIT["metre",1]]],

## PRIMEM["Greenwich",0,

## ANGLEUNIT["degree",0.0174532925199433]],

## CS[ellipsoidal,2],

## AXIS["geodetic latitude (Lat)",north,

## ORDER[1],

## ANGLEUNIT["degree",0.0174532925199433]],

## AXIS["geodetic longitude (Lon)",east,

## ORDER[2],

## ANGLEUNIT["degree",0.0174532925199433]],

## USAGE[

## SCOPE["unknown"],

## AREA["World"],

## BBOX[-90,-180,90,180]],

## ID["EPSG",4326]]

where a $ method allows for retrieving the epsg and proj4string  
values:

x$epsg

## [1] 4326

x$proj4string

## [1] "+proj=longlat +datum=WGS84 +no\_defs"

but this means that packages that hard-code for instance

x[["proj4string"]]

## NULL

now fail to get the result wanted; NULL is not a value that would have  
occurred in legacy code.

Regretably, assignment to a crs object component still works, as the  
objects are lists, so not all downstream legacy code will fail

x$proj4string <- "+proj=longlat +ellps=intl"

x$proj4string

## Warning in `$.crs`(x, proj4string): old-style crs object found: please update

## code

## [1] "+proj=longlat +ellps=intl +no\_defs"

Package maintainers and authors of production scripts will need to  
review their use of crs objects.

Many external data sources provide a WKT CRS directly and as such do not  
have an “input” field. In such cases, the input field is filled with  
the CRS *name*, which is a user-readable representation

st = stars::read\_stars(system.file("tif/L7\_ETMs.tif", package = "stars"))

st\_crs(st)$input

## [1] "UTM Zone 25, Southern Hemisphere"

but this representation can not be used as *input* to a CRS:

st\_crs(st\_crs(st)$input)

## Error in st\_crs.character(st\_crs(st)$input): invalid crs: UTM Zone 25, Southern Hemisphere

however wkt fields obviously *can* be used as input:

st\_crs(st\_crs(st)$wkt) == st\_crs(st)

## [1] TRUE

**CRS objects in sp**

When equiped with a new ( >= 1.5.6) rgdal version, sp’s CRS  
objects carry a comment field with the WKT representation of a CRS:

# install.packages("rgdal", repos="<http://R-Forge.R-project.org>")

library(sp)

(x = CRS("+init=epsg:4326")) # or better: CRS(SRS\_string='EPSG:4326')

## CRS arguments: +proj=longlat +datum=WGS84 +no\_defs

cat(comment(x), "\n")

## GEOGCRS["WGS 84",

## DATUM["World Geodetic System 1984",

## ELLIPSOID["WGS 84",6378137,298.257223563,

## LENGTHUNIT["metre",1]],

## ID["EPSG",6326]],

## PRIMEM["Greenwich",0,

## ANGLEUNIT["degree",0.0174532925199433],

## ID["EPSG",8901]],

## CS[ellipsoidal,2],

## AXIS["longitude",east,

## ORDER[1],

## ANGLEUNIT["degree",0.0174532925199433,

## ID["EPSG",9122]]],

## AXIS["latitude",north,

## ORDER[2],

## ANGLEUNIT["degree",0.0174532925199433,

## ID["EPSG",9122]]],

## USAGE[

## SCOPE["unknown"],

## AREA["World"],

## BBOX[-90,-180,90,180]]]

and it is this WKT representation that is used to communicate with GDAL  
and PROJ when using packages rgdal or sf. At present, rgdal  
generates many warnings about discarded PROJ string keys, intended to  
alert package maintainers and script authors to the need to review code.  
It is particularly egregious to assign to the CRS object projargs  
slot directly, and this is unfortunately seem in much code in packages.

**Coercion from CRS objects to crs and back**

Because workflows often need to combine packages using sp and sf  
representations, coercion methods from CRS to crs have been updated  
to use the WKT information; from sp to sf one can use

(x2 <- st\_crs(x))

## Coordinate Reference System:

## User input: WGS 84

## wkt:

## GEOGCRS["WGS 84",

## DATUM["World Geodetic System 1984",

## ELLIPSOID["WGS 84",6378137,298.257223563,

## LENGTHUNIT["metre",1]],

## ID["EPSG",6326]],

## PRIMEM["Greenwich",0,

## ANGLEUNIT["degree",0.0174532925199433],

## ID["EPSG",8901]],

## CS[ellipsoidal,2],

## AXIS["longitude",east,

## ORDER[1],

## ANGLEUNIT["degree",0.0174532925199433,

## ID["EPSG",9122]]],

## AXIS["latitude",north,

## ORDER[2],

## ANGLEUNIT["degree",0.0174532925199433,

## ID["EPSG",9122]]],

## USAGE[

## SCOPE["unknown"],

## AREA["World"],

## BBOX[-90,-180,90,180]]]

The sp CRS constructor has been provided with an additional argument  
SRS\_string= which accepts WKT, among other representations

(x3 <- CRS(SRS\_string = x2$wkt))

## CRS arguments: +proj=longlat +datum=WGS84 +no\_defs

but also

(x4 <- as(x2, "CRS"))

## CRS arguments: +proj=longlat +datum=WGS84 +no\_defs

uses the WKT information when present.

all.equal(x, x3)

## [1] TRUE

all.equal(x, x4)

## [1] TRUE

**Axis order**

R-spatial packages have, for the past 25 years, pretty much assumed that  
two-dimensional data are XY-ordered, or longitude-latitude. Geodesists,  
on the other hand, typically use \((\phi,\lambda)\), or  
latitude-longitude, as coordinate pairs; the PROJ logo is now  
PR\(\phi\)J. If we use geocentric coordinates, there is no logical  
ordering. Axis direction may also vary; the y-axis index of images  
typically increases when going south., there are  
powers out there that will bring us spatial data with  
(latitude,longitude) as (X,Y) coordinates. Even stronger, *officially*,  
EPSG:4326 has axis order latitude, longitude (see WKT description  
above).

Package sf by default uses a switch in GDAL that brings everything in  
the old, longitude-latitude order.

This can now be controlled (to some extent), as st\_axis\_order can be  
used to query, and set whether axis ordering is “GIS style”  
(longitude,latitude; non-authority compliant) or “authority compliant”  
(often: latitude,longitude):

pt = st\_sfc(st\_point(c(0, 60)), crs = 4326)

st\_axis\_order() # query default: FALSE means interpret pt as (longitude latitude)

## [1] FALSE

st\_transform(pt, 3857)[[1]]

## POINT (0 8399738)

(old\_value = st\_axis\_order(TRUE))

## [1] FALSE

# now interpret pt as (latitude longitude), as EPSG:4326 prescribes:

st\_axis\_order() # query current value

## [1] TRUE

st\_transform(pt, 3857)[[1]]

## POINT (6679169 0)

st\_axis\_order(old\_value) # set back to old value

sf::plot is sensitive to this and will swap axis if needed, but for  
instance ggplot2::geom\_sf is not yet aware of this.

Workflows using sp/rgdal should expect “GIS style” axis order to be  
preserved

rgdal::get\_enforce\_xy()

## [1] TRUE

pt\_sp <- as(pt, "Spatial")

coordinates(pt\_sp)

## coords.x1 coords.x2

## [1,] 0 60

coordinates(spTransform(pt\_sp, CRS(SRS\_string="EPSG:3857")))

## Warning in showSRID(SRS\_string, format = "PROJ", multiline = "NO"): Discarded

## ellps WGS 84 in CRS definition: +proj=merc +a=6378137 +b=6378137 +lat\_ts=0

## +lon\_0=0 +x\_0=0 +y\_0=0 +k=1 +units=m +nadgrids=@null +wktext +no\_defs

## Warning in showSRID(SRS\_string, format = "PROJ", multiline = "NO"): Discarded

## datum WGS\_1984 in CRS definition

## coords.x1 coords.x2

## [1,] 0 8399738