**C code compile tricks:**

1. Put all C code in the src folder. I put all three functions in one file (MRfxns.c). The C functions need to return SEXP types.
2. (already done, so skip) Add a makevars file to the src folder. I copied makevars from the package mgcv, which is another package that uses C code. (You only need to do this once and I already did it, so for this package, skip this step. If you make another package with C code, you’ll need to add the makevars file).
3. (already done, so skip) Add useDynLib(MRTransmissionModel, .registration = TRUE) to NAMESPACE. (Already there, you don’t need to add it again). This was the tricky part. It's really convenient to have roxygen automatically generate the NAMESPACE, but I had to create the NAMESPACE manually to get the C code to communicate with R. You may notice warnings saying that the existing NAMESPACE was not generated by roxygen2 and will not be overwritten. So, if you add a function to the package or create a new S4 class or add another dependency, you have to manually update the NAMESPACE. I can help with this if you decide to make any of those changes.
4. When you build the package (see Quick Notes for how to build), it will automatically update the MRTransmissionModel\_init.c file, and everything \*should\* work beautifully!

**Document an R Function**

1. Click your cursor to inside the name of the function.
2. Go to Code 🡪 Insert Roxygen Skeleton. Click Insert Roxygen Skeleton.
3. Provide a title.
4. Provide a description for each @param
   1. If you add/delete parameters from the function, update the @param documentation
5. Provide a description of what the function returns in @return
6. You may delete @examples.
7. Delete @export if you don’t want the user to be able to directly use this function. Leave @export for user interface functions.
8. Good Practices
   1. Generally we have one function per R script.
   2. Save the function in a script with the same name and .R extension.
   3. Place the script in the R/ folder.

**Document an S4 Class**

1. Click your cursor to inside the name of the class.
2. Go to Code 🡪 Insert Roxygen Skeleton. Click Insert Roxygen Skeleton.
3. Provide a title.
4. If desired, provide additional descriptions for each @slot
5. Delete @return
6. Delete @examples
7. Leave @export
8. Add @docType class
9. Add @rdname name-class (\*see example below)
10. If the class includes the contains field in which it contains subclasses, be sure you set the classes in order, so that classes come after subclasses they contain (\*see example below)

**Document an S4 Method**

1. I put all related methods in the same R script (for example, next.ID.state methods, run methods, etc.)
2. Click your cursor to inside the name of the method.
3. Go to Code 🡪 Insert Roxygen Skeleton. Click Insert Roxygen Skeleton.
4. Provide a title.
5. Add @importFrom is
6. Add @include setClasses.R (this is to link methods to classes)
7. Provide a description of what the methods returns in @return
8. Leave @export
9. Add @docType methods
10. Add @rdname name-methods (\*see example below)

**Add another plot() method**

1. See plot.sim.results.MSIRV.R, plot.sim.results.SIR.R, and plot.sim.results.MSIRV.space.R for examples
2. I put each plot method in a separate script and documented it just like and R function
3. Then add @method plot sim.results.MSIRV (other suitable name)
4. See example below

**Quick Notes**

* Look in the R/ folder for documentation for functions, S4 classes (setClasses.R), S4 methods (run.R, next.ID.state.R), and S3 methods (plot.sim.results.SIR.R, etc.)
* Do not edit anything in the man/ folder
* Add additional data to the data/ folder if it can be saved as a .rda extension
* Add any other data types to the inst/ folder
* Add additional C code to the src/MRfxns.c file
* If you add any functions, classes, S3 methods, or package dependencies, you will need to manually update the NAMESPACE file. See the NAMESPACE file for examples and follow the format given there.
* To build the package run:
  + devtools::document()
  + devtools::build()
  + devtools::check()
* Address all warnings and errors produced by devtools::check()
* Addressing notes produced by devtools::check() is optional and only necessary if you wish to publish the package on CRAN

**Example documentation for S4 class with no subclasses**

#' Class definition for space.nMx object - aka Age Specific Death Rates (ASDR) - by space

#'

#' @slot rate.years numeric; year associated with each time-specific rate

#' @slot rates data.frame; age and space specific death rates per 1 (rows) by year (columns)

#' @slot mid.age numeric; mid-age associated with each age-specific rates

#' @slot n.subpops numeric; number of subpopulations

#'

#' @export

#' @docType class

#' @rdname space.nMx-class

setClass("space.nMx",

representation(rate.years="numeric", #year associated with each time-specific rate

rates="data.frame", #age and space specific death rates per 1 (rows) by year (columns)

mid.age="numeric", #mid-age associated with each age-specific rates

n.subpops="numeric" #number of subpopulations

))

**Example documentation for S4 class with 3 subclasses**

#' Generate of object of class experiment result (sim.result)

#'

#' Class to hold SIR simulation results. Hold the results of the simulation as

#' SIR compartment values at each time step.

#'

#' @slot .Data matrix.

#' @slot s.inds numeric.

#' @slot i.inds numeric.

#' @slot r.inds numeric.

#' @slot age.class numeric.

#' @slot t numeric.

#'

#' @export

#' @docType class

#' @rdname sim.results.SIR-class

setClass("**sim.results.SIR**",

slots = list(.Data = "matrix",

s.inds = "numeric", #indexing of susceptibles

i.inds = "numeric", #infectious indexes

r.inds = "numeric", #recovered indexes

age.class = "numeric", #upper end of age classes

t = "numeric"#times when we have info in the matrix

),

contains="matrix") **# you don’t have to set the class matrix, so we start with this class**

#' Holds the results of a simulation with an MSIRV object

#'

#' @slot m.inds numeric.

#' @slot v.inds numeric.

#' @slot routine.intro numeric.

#' @slot sia.times numeric.

#'

#' @export

#' @docType class

#' @rdname sim.results.MSIRV-class

setClass("**sim.results.MSIRV**",

slots = list(m.inds = "numeric",

v.inds = "numeric",

routine.intro = "numeric",

sia.times = "numeric"),

**contains= "sim.results.SIR"**) # contains the class above

#' Holds the results of a simulation with an experiment.updatedemog object

#'

#' @slot births.each.timestep numeric.

#' @slot growth.rate.each.timestep numeric.

#'

#' @export

#' @docType class

#' @rdname sim.results.MSIRV.update.demog-class

setClass("**sim.results.MSIRV.update.demog**",

slots = list(births.each.timestep = "ANY",

growth.rate.each.timestep = "ANY"

),

**contains="sim.results.MSIRV")** # contains the class above

#' Holds the results output for a simulation with an experiment.updatedemog.vaccinationchange object

#'

#' @slot MR1.fail.each.timestep numeric.

#' @slot MR2.fail.each.timestep numeric.

#' @slot SIA.fail.each.timestep numeric.

#'

#' @export

#' @docType class

#' @rdname sim.results.MSIRV.update.demog.vaccine.change-class

setClass(**"sim.results.MSIRV.update.demog.vaccine.change",**

slots = list(MR1.fail.each.timestep = "ANY",

MR2.fail.each.timestep = "ANY",

SIA.fail.each.timestep = "ANY"

),

**contains="sim.results.MSIRV.update.demog")** # contains the class above

**Example documentation for S4 methods**

#' Methods for Getting State at Next Time Step (next.ID.state methods)

#'

#' @param state current state system (ID.state.matric)

#' @param tran ID transition object that has all info for supervising the transition

#' @param ... ignored

#'

#' @importFrom methods is

#' @include setClasses.R

#' @return an updated ID.state.matrix

#' @export

#' @docType methods

#' @rdname next.ID.state-methods

setGeneric("next.ID.state",

function(state, tran, ...) standardGeneric("next.ID.state"))

#' @rdname next.ID.state-methods

#' @aliases next.ID.state,ID.state.matrix,ID.transition.SIR-method

setMethod("next.ID.state",

c("ID.state.matrix", "ID.transition.SIR"),

SIRIDTran)

#' @rdname next.ID.state-methods

#' @aliases next.ID.state,ID.state.matrix,ID.transition.SIR.vac-method

setMethod("next.ID.state",

c("ID.state.matrix", "ID.transition.SIR.vac"),

function(state, tran) {

#A bit of a hack, so we do not double deal with vaccination

if (!is(tran, "ID.transition.MSIRV")) {

delta.state <- state[tran@s.inds]\*

(tran@vac.per@pvacc.in.age.class)

state[tran@s.inds] <- state[tran@s.inds] - delta.state

state[tran@r.inds] <- state[tran@r.inds] + delta.state

mid.age.class <- (tran@age.class + c(0, tran@age.class[2:tran@n.age.class-1]))/2

sia.sv.prob <-tran@sia.vac \* pvacsuccess(mid.age.class, tran@sia.vsucc)

delta.sia <- state[tran@s.inds]\*sia.sv.prob

state[tran@s.inds] <- state[tran@s.inds] - delta.sia

state[tran@r.inds] <- state[tran@r.inds] + delta.sia

}

rc <- SIRIDTran(state,tran) #avoid callNextMethod overhead

return(rc)

})

**Example documentation for plot methods**

#' Plot method for class "sim.results.MSIRV"

#'

#' @param x object of class "sim.results.MSIRV"

#' @param ... ignored

#'

#' @importFrom graphics par lines

#' @importFrom grDevices rgb col2rgb

#'

#' @return plot object

#' @method plot sim.results.MSIRV

#' @export

plot.sim.results.MSIRV <- function (x,...) {

# plotting parameters

from=0

to=max(x@t)

low.age = 0

high.age = max(x@age.class)

proportions = FALSE

plot.events = TRUE

par(mfcol = c(4,1))

par(mar=c(3,4,2,2))

orig.t <- x@t

x@.Data <- x[,x@t>=from & x@t<=to]

x@t <- x@t[x@t>=from & x@t<=to]

age.classes <- c()

for (i in 1:length(x@age.class)) {

age.classes <- c(age.classes, rep(x@age.class[i],5))

}

#make everything be in the age classes of interest

x@.Data <- x[age.classes>=low.age & age.classes<=high.age,]

x@m.inds <- x@m.inds[x@age.class>=low.age & x@age.class<=high.age]

x@s.inds <- x@s.inds[x@age.class>=low.age & x@age.class<=high.age]

x@i.inds <- x@i.inds[x@age.class>=low.age & x@age.class<=high.age]

x@r.inds <- x@r.inds[x@age.class>=low.age & x@age.class<=high.age]

x@v.inds <- x@v.inds[x@age.class>=low.age & x@age.class<=high.age]

x@age.class <- x@age.class[x@age.class>=low.age & x@age.class<=high.age]

tots <- getCompartmentTotals(x)

if (proportions) {

tots <- toProportions(tots)

}

plt.events <- function() {

if (!plot.events) return()

sia.times <- orig.t[which(x@sia.times>0)]

routine.times <- orig.t[which(x@routine.intro>0)]

if (length(sia.times)>0) {

for (i in 1:length(sia.times)) {

lines(rep(sia.times[i],2),c(0, 10^10),

col=rgb(t(col2rgb("cornflowerblue")/256), alpha=.45),

lty=3)

}

}

if (length(routine.times)>0) {

for (i in 1:length(routine.times)) {

lines(rep(routine.times[i],2),c(0, 10^10),

col=rgb(t(col2rgb("coral")/256), alpha=.45),

lty=3)

}

}

}

plot(tots@t, tots[tots@s.inds,]+tots[tots@m.inds,], type="b", xlab="t",

ylab="M+S", cex=.75, ylim=c(0, max(tots[tots@s.inds,]+tots[tots@m.inds,])))

lines(tots@t, tots[tots@s.inds,], lty=2, col="green")

lines(tots@t, tots[tots@m.inds,], lty=3, col="blue")

plt.events()

plot(tots@t,tots[tots@i.inds,], type="b", xlab="t", ylab="I", cex=.75)

plt.events()

plot(tots@t,tots[tots@r.inds,]+tots[tots@v.inds,], type="b", xlab="t",

ylab="R+V", cex=.75,ylim=c(0, max(tots[tots@r.inds,]+tots[tots@v.inds,])))

lines(tots@t, tots[tots@r.inds,], lty=2, col="green")

lines(tots@t, tots[tots@v.inds,], lty=3, col="blue")

plt.events()

plot(x@t,getAverageInfectionAge(x) , type="b", xlab="t", ylab="Age", cex=.75)

plt.events()

}