1. Overview of sftrack

This vignette acts as a proto how-to manually mainly for highlighting features for anyone testing sftrack. It will eventually evolve into the full vignette.

We'll begin with a brief overview of an `sftrack` object. An `sftrack` object is a data object that describes the movement of a subject, it has x,y coordinates (sometimes z), some measurement of time (clock time or a sequence of integers), and some grouping variable that identifies the subjects. For the spatial aspects of `sftrack` we are using the package `sf`, this powerful tool lets us quickly calculate spatial attributes and plot with ease with its full integration with `rgdal`.

An `sftrack` object has 4 parts to it, 3 of which are required:
- **Geometry** This is stored as an `sfc` object from `sf`. Accepts x,y,z coordinates, although z is rarely used.
- **Burst** This is the grouping variables, which contains at minimum an id field to identify the subject.
- **Time** This can either be a POSIX object or an integer.
- **Error** (optional). This is the field with error data in it.

Loading in raw data

To create `sftrack` objects data we use the `as_sftrack()` or `as_sftraj()` function, depending on your desired output. Both have the same arguments and output but differ in the way the geometry field is calculated.

`as_sftrack()` accepts 2 kinds of raw data for each of the 4 required parts. Either a vector/list representing the data where length = nrow(data), or it accepts the column name where the data exists. For any `sftrack` component you can input either vector data or the column name, but not both.

Global options

These are options that are required regardless of which input type you use.

data - is a data.frame containing your data. At present we are reserving 'burst' as a column name, so data will be overwritten if this column name exists.
crs - the coordinate references system/projection of the data, as implemented by rgdal. see CRS-class for more information. If none is supplied crs is set as NA and can be set later.
active_burst - This is a vector containing what bursts are 'active'. Meaning calculations and graphing will be grouped by these bursts. Can change active_burst whenever. If no value is supplied it defaults to all bursts.

Vector inputs

Vector inputs to `as_sftrack` in general involve feeding as_sftrack the data itself where length(vector) == nrow(data). Or a list where each component adheres to this rule.

burst_list - a list with named vectors to group the sftrack where each list item is length(vector) = nrow(data). One item must be named `id`, but otherwise can be infinite number of grouping variables.
xyz - data.frame of x,y,z coordinates where column with order : `c(x, y, z)`, z is optional. NAs are allowed, alhough NAs must exist through the entire row otherwise an error is thrown.
time - a vector containing the time information, must be either POSIX or an integer where length(time) == nrow(data). Using this argument will name the time column as 'reloc_time'.
error - a vector containing the error information where length(error) == nrow(data). Using this argument will name the error information as 'track_error'. Input can be singular NA, inwhich the column is filled with NAs.

Examples (Vector)

```
raccoon_data <- read.csv(system.file('extdata/raccoon_data.csv', package='sftrack'))

#data
data = raccoon_data
#xyz
xyz = data[,c('longitude','latitude')]
crs = '+init=epsg:4326'
#bursts
burst_list = list(id = raccoon_data$sensor_code,month = as.POSIXlt(raccoon_data$utc_date)$mon+1)
active_burst = c('id','month')
#time
time = as.POSIXct(raccoon_data$acquisition_time, tz='EST')
#error
error = data$fix
my_sftrack <- as_sftrack(data = data, xyz = xyz, burst_list = burst_list,
                         active_burst = active_burst, time = time,
                         crs = crs, error = error)

head(my_sftrack)
```

```
## This is an sftrack object
## crs: +init=epsg:4326
## bursts : total = 2 | active burst = id, month
## Rows: 6 | Cols: 14
##   sensor_code   utc_date utc_time latitude longitude height hdop vdop fix
## 1        CJ11 2019-01-19 00:02:30       NA        NA     NA  0.0  0.0  NO
## 2        CJ11 2019-01-19 01:02:30 26.06945 -80.27906      7  6.2  3.2  2D
## 3        CJ11 2019-01-19 02:02:30       NA        NA     NA  0.0  0.0  NO
## 4        CJ11 2019-01-19 03:02:30       NA        NA     NA  0.0  0.0  NO
## 5        CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2  2D
## 6        CJ11 2019-01-19 05:02:30 26.06867 -80.27930    350  1.9  3.2  3D
##      acquisition_time          reloc_time sftrack_error
## 1 2019-01-19 00:02:30 2019-01-19 00:02:30            NO
## 2 2019-01-19 01:02:30 2019-01-19 01:02:30            2D
## 3 2019-01-19 02:02:30 2019-01-19 02:02:30            NO
## 4 2019-01-19 03:02:30 2019-01-19 03:02:30            NO
## 5 2019-01-19 04:02:30 2019-01-19 04:02:30            2D
## 6 2019-01-19 05:02:30 2019-01-19 05:02:30            3D
##                   burst                   geometry
## 1 (id: CJ11, month: 1)                POINT EMPTY
## 2 (id: CJ11, month: 1) POINT (-80.27906 26.06945)
## 3 (id: CJ11, month: 1)                POINT EMPTY
## 4 (id: CJ11, month: 1)                POINT EMPTY
## 5 (id: CJ11, month: 1) POINT (-80.27431 26.06769)
## 6 (id: CJ11, month: 1)  POINT (-80.2793 26.06867)
```

As you can see in this case the data is not overwritten, but extra columns added with the correct data.

---

data.frame inputs

Data.frame inputs generally describe the columns in the data that represent each field.

coords - a vector listing the column names for each x,y,z coordinate, just like `sf`, where again position 1 = x, 2 = y, 3 = z. ex: `c('longitude','latitude','altitude')`.
id - a character string naming the column with the id information.
burst_col - a vector with character strings naming additional burst information. This field is not required, and should not contain the `id` field.
time_col - a character string naming the column with the time information. Must be POSIX or integer.
error_col - a character string naming the column with the error information. If NA, error column is stored as NA and not accessible.

Examples (data.frame inputs)

```
data$time <- as.POSIXct(data$acquisition_time, tz='EST')
data$month <- as.POSIXlt(data$acquisition_time)$mon+1

coords = c('longitude','latitude')
id = 'sensor_code'
burst_col = c('month')
time_col = 'time'
error_col = 'fix'

my_sftraj <- as_sftraj(data = data, coords = coords, id = id, burst_col = burst_col, time_col = time_col

head(my_sftraj)
```

```
## This is an sftraj object
## crs: NA
## bursts : total = 2 | active burst = id, month
## Rows: 6 | Cols: 14
##   sensor_code   utc_date utc_time latitude longitude height hdop vdop fix
## 1        CJ11 2019-01-19 00:02:30       NA        NA     NA  0.0  0.0  NO
## 2        CJ11 2019-01-19 01:02:30 26.06945 -80.27906      7  6.2  3.2  2D
## 3        CJ11 2019-01-19 02:02:30       NA        NA     NA  0.0  0.0  NO
## 4        CJ11 2019-01-19 03:02:30       NA        NA     NA  0.0  0.0  NO
## 5        CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2  2D
## 6        CJ11 2019-01-19 05:02:30 26.06867 -80.27930    350  1.9  3.2  3D
##      acquisition_time                time month                burst
## 1 2019-01-19 00:02:30 2019-01-19 00:02:30     1 (id: CJ11, month: 1)
## 2 2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11, month: 1)
## 3 2019-01-19 02:02:30 2019-01-19 02:02:30     1 (id: CJ11, month: 1)
## 4 2019-01-19 03:02:30 2019-01-19 03:02:30     1 (id: CJ11, month: 1)
## 5 2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11, month: 1)
## 6 2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11, month: 1)
##                          geometry
## 1 GEOMETRYCOLLECTION (POINT E...
## 2 GEOMETRYCOLLECTION (POINT (...
## 3 GEOMETRYCOLLECTION (POINT E...
## 4 GEOMETRYCOLLECTION (POINT E...
## 5 LINESTRING (-80.27431 26.06...
## 6 LINESTRING (-80.2793 26.068...
```

Conversion mode

`as_sftrack()` and `as_sftraj()` also accept other data types, and the arguments differ depending on the class. It currenly accepts, `sf`, `ltraj`, and eventually `tibbles`.

Import from ltraj

For an ltraj all you need is the ltraj object, all relevant information is taken from the object.

```
library(adehabitatLT)

ltraj_df <- as.ltraj(xy=raccoon_data[,c('longitude','latitude')], date = as.POSIXct(raccoon_data$acquis
 id = raccoon_data$sensor_code, typeII = TRUE,
 infolocs = raccoon_data[,1:6] )

my_sf <- as_sftrack(ltraj_df)
head(my_sf)
```

```
## This is an sftrack object
## crs: NA
## bursts : total = 1 | active burst = id
## Rows: 6 | Cols: 11
##           x        y          reloc_time sensor_code   utc_date utc_time
## 1        NA       NA 2019-01-19 00:02:30        CJ11 2019-01-19 00:02:30
## 2 -80.27906 26.06945 2019-01-19 01:02:30        CJ11 2019-01-19 01:02:30
## 3        NA       NA 2019-01-19 02:02:30        CJ11 2019-01-19 02:02:30
## 4        NA       NA 2019-01-19 03:02:30        CJ11 2019-01-19 03:02:30
## 5 -80.27431 26.06769 2019-01-19 04:02:30        CJ11 2019-01-19 04:02:30
## 6 -80.27930 26.06867 2019-01-19 05:02:30        CJ11 2019-01-19 05:02:30
##   latitude longitude height      burst                   geometry
## 1       NA        NA     NA (id: CJ11)                 POINT EMPTY
## 2 26.06945 -80.27906      7 (id: CJ11) POINT (-80.27906 26.06945)
## 3       NA        NA     NA (id: CJ11)                 POINT EMPTY
## 4       NA        NA     NA (id: CJ11)                 POINT EMPTY
## 5 26.06769 -80.27431    858 (id: CJ11) POINT (-80.27431 26.06769)
## 6 26.06867 -80.27930    350 (id: CJ11)  POINT (-80.2793 26.06867)
```

sf objects

`sf` objects are handled similiarly to the standard raw data, except you do not need to input any information about the coordinates or projection.

```
library(sf)
```

```
## Linking to GEOS 3.7.0, GDAL 2.4.0, PROJ 5.2.0
```

```
df1 <- data[!is.na(raccoon_data$latitude),]
sf_df <- st_as_sf(df1, coords=c('longitude','latitude'), crs = crs)
id = 'sensor_code'
time_col = 'time'

new_sftraj <- as_sftraj(sf_df,id=id, time_col = time_col)
head(new_sftraj)
```

```
## This is an sftraj object
## crs: +init=epsg:4326
## bursts : total = 1 | active burst = id
## Rows: 6 | Cols: 12
##    sensor_code   utc_date utc_time height hdop vdop fix
## 2         CJ11 2019-01-19 01:02:30      7  6.2  3.2  2D
## 5         CJ11 2019-01-19 04:02:30    858  5.1  3.2  2D
## 6         CJ11 2019-01-19 05:02:30    350  1.9  3.2  3D
## 7         CJ11 2019-01-19 06:02:30     11  2.3  4.5  3D
## 8         CJ11 2019-01-19 07:02:04      9  2.7  3.9  3D
## 10        CJ11 2019-01-19 17:02:30     NA  2.0  3.3  3D
##       acquisition_time                time month      burst
## 2  2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11)
## 5  2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11)
## 6  2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11)
## 7  2019-01-19 06:02:30 2019-01-19 06:02:30     1 (id: CJ11)
## 8  2019-01-19 07:02:04 2019-01-19 07:02:04     1 (id: CJ11)
## 10 2019-01-19 17:02:30 2019-01-19 17:02:30     1 (id: CJ11)
##                           geometry
## 2  LINESTRING (-80.27906 26.06...
## 5  LINESTRING (-80.27431 26.06...
## 6  LINESTRING (-80.2793 26.068...
## 7  LINESTRING (-80.27908 26.06...
## 8  LINESTRING (-80.27902 26.06...
## 10 LINESTRING (-80.279 26.0698...
```

```
new_sftrack <- as_sftrack(sf_df, id=id, time_col = time_col)
head(new_sftrack)
```

```
## This is an sftrack object
## crs: +init=epsg:4326
## bursts : total = 1 | active burst = id
## Rows: 6 | Cols: 12
##    sensor_code   utc_date utc_time height hdop vdop fix
## 2         CJ11 2019-01-19 01:02:30      7  6.2  3.2  2D
## 5         CJ11 2019-01-19 04:02:30    858  5.1  3.2  2D
## 6         CJ11 2019-01-19 05:02:30    350  1.9  3.2  3D
## 7         CJ11 2019-01-19 06:02:30     11  2.3  4.5  3D
## 8         CJ11 2019-01-19 07:02:04      9  2.7  3.9  3D
## 10        CJ11 2019-01-19 17:02:30     NA  2.0  3.3  3D
##       acquisition_time                time month      burst
## 2  2019-01-19 01:02:30 2019-01-19 01:02:30     1 (id: CJ11)
## 5  2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11)
## 6  2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11)
## 7  2019-01-19 06:02:30 2019-01-19 06:02:30     1 (id: CJ11)
## 8  2019-01-19 07:02:04 2019-01-19 07:02:04     1 (id: CJ11)
## 10 2019-01-19 17:02:30 2019-01-19 17:02:30     1 (id: CJ11)
##                        geometry
## 2  POINT (-80.27906 26.06945)
## 5  POINT (-80.27431 26.06769)
## 6   POINT (-80.2793 26.06867)
## 7  POINT (-80.27908 26.06962)
## 8  POINT (-80.27902 26.06963)
## 10   POINT (-80.279 26.06982)
```

Inter-class conversion

Additionally `as_sftrack` and `as_sftraj` can convert back and forth between each other.

```
# Make tracks from raw data
my_sftrack <- as_sftrack(data = data, coords = coords, id = id, burst_col = burst_col, time_col = time_
my_sftraj <- as_sftraj(data = data, coords = coords, id = id, burst_col = burst_col, time_col = time_co

# Convert between types
new_sftrack <- as_sftrack(my_sftraj)
#head(new_sftrack)
new_sftraj <- as_sftraj(my_sftrack)
#head(new_sftraj)

all.equal(my_sftraj,new_sftraj)
```

```
## [1] TRUE
```

```
all.equal(my_sftrack,new_sftrack)
```

```
## [1] TRUE
```

Some basic functionality of sf_track and sf_traj objects

print

`print()` prints out the type of object as well as specific data on the sf_track object. Additionally you can supply the number of rows or columns you'd like to display with arguments `n_row` and `n_col`. When using `n_col` the display will show the `burst` and `geometery` fields as well as any other columns starting from column 1 until `#columns + 2 = n_col`. If neither is provided than print just uses default values in the global options. `ncol` and `nrow` are optional arguments, defaults to data.frame defaults.

```
print(my_sftrack,5,10)
```

```
## This is an sftrack object
## crs: NA
## bursts : total = 2 | active burst = id, month
## Rows: 445 | Cols: 14
##   sensor_code   utc_date utc_time latitude longitude height hdop vdop ...
## 1        CJ11 2019-01-19 00:02:30       NA        NA     NA  0.0  0.0 ...
## 2        CJ11 2019-01-19 01:02:30 26.06945 -80.27906      7  6.2  3.2 ...
## 3        CJ11 2019-01-19 02:02:30       NA        NA     NA  0.0  0.0 ...
## 4        CJ11 2019-01-19 03:02:30       NA        NA     NA  0.0  0.0 ...
## 5        CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2 ...
##                  burst                  geometry
## 1 (id: CJ11, month: 1)                POINT EMPTY
## 2 (id: CJ11, month: 1) POINT (-80.27906 26.06945)
## 3 (id: CJ11, month: 1)                POINT EMPTY
## 4 (id: CJ11, month: 1)                POINT EMPTY
## 5 (id: CJ11, month: 1) POINT (-80.27431 26.06769)
```

summary

`summary()` works as youd normally expect for a data.frame, except it displays the burst column as a count of each active_burst combination.

```
summary(my_sftrack)
```

```
##  sensor_code       utc_date       utc_time      latitude
##  CJ11:222    2019-01-19: 32   17:02:30: 26   Min.   :26.07
##  CJ13:223    2019-01-20: 32   23:02:30: 20   1st Qu.:26.07
##              2019-01-21: 32   00:02:30: 19   Median :26.07
##              2019-01-22: 32   18:02:30: 19   Mean   :26.07
##              2019-01-23: 32   01:02:30: 17   3rd Qu.:26.07
##              2019-01-25: 32   07:02:30: 17   Max.   :26.08
##              (Other)   :253   (Other) :327   NA's   :168
##    longitude          height            hdop           vdop
##  Min.   :-80.28   Min.   : -30.00   Min.   :0.000   Min.   :0.000
##  1st Qu.:-80.28   1st Qu.:   1.00   1st Qu.:0.000   1st Qu.:0.000
##  Median :-80.28   Median :   7.00   Median :1.300   Median :1.900
##  Mean   :-80.28   Mean   :  36.65   Mean   :1.691   Mean   :1.938
##  3rd Qu.:-80.28   3rd Qu.:  15.50   3rd Qu.:2.500   3rd Qu.:3.200
##  Max.   :-80.27   Max.   :1107.00   Max.   :9.900   Max.   :8.400
##  NA's   :168      NA's   :198
##  fix                acquisition_time      time
##  2D: 37   2019-01-19 00:02:30:  2     Min.   :2019-01-19 00:02:30
##  3D:240   2019-01-19 01:02:30:  2     1st Qu.:2019-01-22 07:02:30
##  NO:168   2019-01-19 04:02:30:  2     Median :2019-01-25 23:02:30
##           2019-01-19 06:02:30:  2     Mean   :2019-01-25 22:22:18
##           2019-01-19 17:02:30:  2     3rd Qu.:2019-01-29 07:02:09
##           2019-01-20 02:02:30:  2     Max.   :2019-02-01 23:02:30
##           (Other)            :433
##      month          burst        geometry
##  Min.   :1.000   CJ11_1:207   POINT  :445
##  1st Qu.:1.000   CJ11_2: 15   epsg:NA:  0
##  Median :1.000   CJ13_1:208
##  Mean   :1.067   CJ13_2: 15
##  3rd Qu.:1.000
##  Max.   :2.000
##
```

summary_sftrack

`summary_sftrack()` is a special summary function specific for sftrack objects. It summarizes the data based on the beginning and end of each burst as well as the total distance of the burst. This function uses `st_length` from the `sf` package and therefore outputs in units of the crs. In this example the distance is in degrees distance.

```
summary_sftrack(my_sftrack)
```

```
##        points          begin_time            end_time       length
## CJ11_1    207 2019-01-19 00:02:30 2019-01-31 23:02:30 0.2285805244
## CJ11_2     15 2019-02-01 00:02:30 2019-02-01 23:02:30 0.0181846139
## CJ13_1    208 2019-01-19 00:02:30 2019-01-31 23:02:30 0.0949271187
## CJ13_2     15 2019-02-01 00:02:30 2019-02-01 23:02:07 0.0003190602
```

You can also trigger this function by using `summary(data, stats = TRUE)`

```
summary(my_sftrack, stats = TRUE)
```

```
##        points          begin_time            end_time       length
```

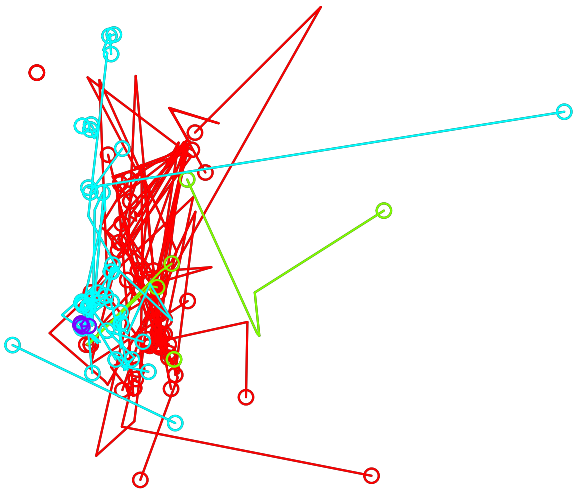
```
## CJ11_1    207 2019-01-19 00:02:30 2019-01-31 23:02:30 0.2285805244
## CJ11_2     15 2019-02-01 00:02:30 2019-02-01 23:02:30 0.0181846139
## CJ13_1    208 2019-01-19 00:02:30 2019-01-31 23:02:30 0.0949271187
## CJ13_2     15 2019-02-01 00:02:30 2019-02-01 23:02:07 0.0003190602
```

Plotting

Base plotting

Currently there are some basic plotting methods. Base plotting currently does not have any thrills built into it, and assumes that the `active_burst` is the grouping/coloring variable.

```
plot(my_sftraj)
```

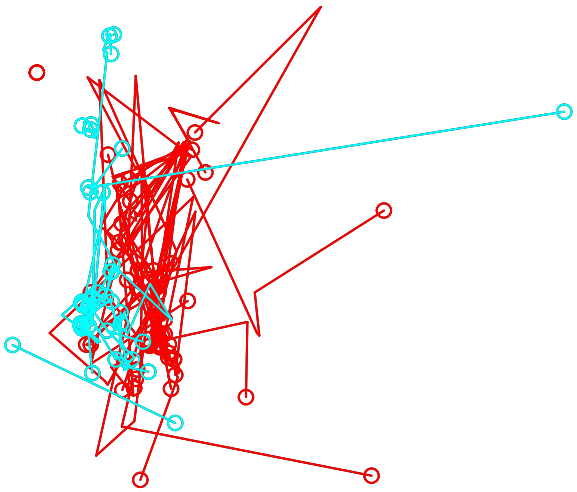


And changing the active burst will change the plot view

```
active_burst(my_sftraj$burst) <- 'id'
active_burst(my_sftraj$burst)
```

```
## [1] "id"
```

```
plot(my_sftraj)
```



ggplot

This is a work in progress, but theres a rudimentary geom_sftrack function. As of now you have to input `data` into the geom_sftrack function. That'll change as I look more into it. Again ggplot assumes active_burst is the grouping variable. Plots vary slightly based on if they're track of traj

```
library(ggplot2)
ggplot() + geom_sftrack(data = my_sftraj)
```

Bursts

Bursts are a big emphasis in the class. They are made in a similar vein to the `sfc` and `sfg` in `sf`. `ind_burst` is a singular burst. Its whats stored at the row level. A `multi-burst` is a collection of `ind_bursts` and exists at the column level. Bursts also have an `active_burst` argument, which turns on and off certain bursts for analysis and plotting purposes.

We can look at the structure

```
mb1 <- make_multi_burst(burst_list=burst_list, active_burst=c('id','month'))
str(mb1)
```

```
## multi_burst of length 445; first list element: List of 2
##  $ id   : chr "CJ11"
##  $ month: chr "1"
##  - attr(*, "class")= chr "ind_burst"
```

```
mb1[[1]]
```

```
## $id
```

```
## [1] "CJ11"
##
## $month
## [1] "1"
```

A burst contains grouping information, where the id of the subject is required, but additional bursts are not. Every burst has the same burst columns available, and these bursts are stored internally as a list. You can also see that labels are created based on the active_burst, this can be accessed and makes for easy labeling of plots of figures.

Basics

ind_bursts

An ind_burst is the grouping variables for a single row of data.

You can make an `ind_burst` object using `make_ind_burst()`, and giving it a list with the bursts named.

```
indb <- make_ind_burst(list(id='CJ13', month = 4))
indb
```

```
## $id
## [1] "CJ13"
##
## $month
## [1] "4"
```

Because ind_bursts are simply lists, you can edit individual elements in an ind_burst

```
indb
```

```
## $id
## [1] "CJ13"
##
## $month
## [1] "4"
```

```
indb[1] <- 'CJ15'
str(indb)
```

```
## List of 2
##  $ id   : chr "CJ15"
##  $ month: chr "4"
##  - attr(*, "class")= chr "ind_burst"
```

```
indb$month <- '5'
str(indb)
```

```
## List of 2
##  $ id   : chr "CJ15"
##  $ month: chr "5"
##  - attr(*, "class")= chr "ind_burst"
```

multi_burst

Multi_bursts are a collection of ind_bursts, where all ind_bursts must have the same grouping variables. Multi_bursts have a specific 'active_burst', which is the bursts that should be activated to group the data when doing calculations, plotting, and graphing.

Similarly to ind_burst you can make a multi_burst with `make_multi_burst()`. The argument `burst_list` takes a list where each element is a vector indicating the named burst as well as a vector of the active bursts.

```
burst_list <- list(id = rep(1:2,10), year = rep(2020, 10))
mb <- make_multi_burst(burst_list=burst_list, active_burst=c('id','year'))
str(mb)
```

```
## multi_burst of length 20; first list element: List of 2
##  $ id  : chr "1"
##  $ year: chr "2020"
##  - attr(*, "class")= chr "ind_burst"
```

You can also make a multi_burst by concatenating multiple ind_bursts. Though in this case the active_burst will default to all bursts. Which you can change later.

```
a <- make_ind_burst(list(id = 1, year = 2020))
b <- make_ind_burst(list(id = 1, year = 2021))
c <- make_ind_burst(list(id = 2, year = 2020))
mb <- c(a, b , c)
```

```
## Warning in check_two_bursts(mb): 1_2020 & 1_2021 & 2_2020 has only one
## relocation
```

```
summary(mb)
```

```
## 1_2020 1_2021 2_2020
##      1      1      1
```

You can also combine multi_bursts together with `c()`.

```
mb_combine <- c(mb1,mb1)
summary(mb_combine)
```

```
## CJ11_1 CJ11_2 CJ13_1 CJ13_2
##    414     30    416     30
```

You can also edit bursts like a list, but you must replace it with an object of the appropriate class

```
mb[1]
```

```
## [[1]]
## $id
## [1] "1"
##
## $year
## [1] "2020"
```

```
mb[1] <- make_ind_burst(list(id=3,year=2019))
```

```
## Warning in check_two_bursts(ret): 1_2021 & 2_2020 & 3_2019 has only one
## relocation
```

```
mb[1]
```

```
## [[1]]
## $id
## [1] "3"
##
## $year
## [1] "2019"
```

And the burst names must match the ones in the multi_burst

```
# Try to add an ind_burst with a month field when the original burst had year instead
tryCatch(mb[1] <- make_ind_burst(list(id=3,month=2019)), error = function(e) e)
```

```
## <simpleError in check_burst_names(ret): Burst names do not match>
```

burst_sort

Multi_burst calculates an index based on the active_burst called the `sort_index`, this is simply a factor of the 'active' bursts where each individual burst = paste(active_burst, sep='_'). As its a factor it can double as the labels and sorting index simultaneously. This gets recalculated everytime a multi_burst is modified or created.

You can use `burst_sort()` to access this value, but you can not modify it.

```
burst_sort(mb1)[1:10]
```

```
##  [1] CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1 CJ11_1
## Levels: CJ11_1 CJ11_2 CJ13_1 CJ13_2
```

burst_levels

On occasion your data may have more possible grouping levels than the sort_index shows. For example if there is a data gap, where the specific level may need to be retained even if no data is associated with it. Because of this we allow burst_levels to be amended to add more levels than are in the current data. Since this variable is a factor, we simply are redefining the factor levels with `burst_levels()`. This may also be useful if you'd like to rearrange the order of plotting.

```
burst_levels(mb1)
```

```
## [1] "CJ11_1" "CJ11_2" "CJ13_1" "CJ13_2"
```

```
burst_levels(mb1) <- c("CJ11_1", "CJ11_2", "CJ13_1" ,"CJ13_2", "CJ14_3")
burst_levels(mb1)
```

```
## [1] "CJ11_1" "CJ11_2" "CJ13_1" "CJ13_2" "CJ14_3"
```

active_burst

The active_burst is a simple yet powerful feature. It dictates how your data is grouped for essentially all calculations. It can also be changed on the fly. You can also view and change the active_burst of a multi_burst with `active_burst()`. This calculates a new sort_index internally as well.

```
active_burst(my_sftrack$burst)
```

```
## [1] "id"    "month"
```

```
summary(my_sftrack, stats = T)
```

```
##        points          begin_time            end_time       length
## CJ11_1    207 2019-01-19 00:02:30 2019-01-31 23:02:30 0.2285805244
## CJ11_2     15 2019-02-01 00:02:30 2019-02-01 23:02:30 0.0181846139
## CJ13_1    208 2019-01-19 00:02:30 2019-01-31 23:02:30 0.0949271187
## CJ13_2     15 2019-02-01 00:02:30 2019-02-01 23:02:07 0.0003190602
```

```
active_burst(my_sftrack$burst) <- c('id')
active_burst(my_sftrack$burst)
```

```
## [1] "id"
```

```
summary(my_sftrack, stats = T)
```

```
##      points          begin_time            end_time     length
## CJ11    222 2019-01-19 00:02:30 2019-02-01 23:02:30 0.24905342
## CJ13    223 2019-01-19 00:02:30 2019-02-01 23:02:07 0.09573788
```

burst_select

Sometimes you may want to access to the active_burst data, especially as a developer. To do this you can `burst_select()` which returns only the active burst data in your burst.

```
burst_select(mb1)[1:3]
```

```
## [[1]]
## [[1]]$id
## [1] "CJ11"
##
## [[1]]$month
## [1] "1"
##
##
## [[2]]
## [[2]]$id
## [1] "CJ11"
##
## [[2]]$month
## [1] "1"
##
##
## [[3]]
## [[3]]$id
## [1] "CJ11"
##
## [[3]]$month
## [1] "1"
```

This is also the easiest way to subset the bursts, you can `select` a new set of bursts which will subset the burst by those new columns, but it will not change the original active_burst.

```
burst_select(mb1, select = 'id')[1:3]
```

```
## [[1]]
## [[1]]$id
## [1] "CJ11"
##
##
## [[2]]
## [[2]]$id
## [1] "CJ11"
##
##
## [[3]]
## [[3]]$id
## [1] "CJ11"
```

There are two ways to access the labels of the bursts. First you can access them as previously mentioned from the sort_index attribute via `burst_sort` and `burst_levels`. These attributes should actively update as the multi_burst is updated, and is a fast an effecient method of subsetting. A second way of accessing them is via the burst itself. `burst_labels` creates the burst labels from the burst when called from each ind_burst. This may be more reliable as it recreates the burst labels from the original burst, but can also be much slower with large datasets than using `burst_sort` as `burst_sort` is already calculated when the burst was created.

`burst_labels` is generally an internal function, as this function is how the original `sort_index` is created. However it may be of interest to developers, if you'd like to manually recalculate the index for example. It can also create novel labels, by supplying a new `active_burst` argument.

```
burst_labels(mb1)[1:3]
```

```
## [1] "CJ11_1" "CJ11_1" "CJ11_1"
```

```
burst_labels(mb1, active_burst = 'id')[1:3]
```

```
## [1] "CJ11" "CJ11" "CJ11"
```

Geometry column

As stated earlier, the geometry column is built using `sf`, so functions exactly as it would in `sf`. You can modify it and redefine it using the `sf` tools. More specifically the geometry column of an sf_track object is a `sfc` column. The main difference between a standard `sf` object created using `st_as_sf` is that we automatically allow empty geometries, where as this option is turned off by default in `st_as_sf()`.

```
my_sftrack$geometry
```

```
## Geometry set for 445 features  (with 168 geometries empty)
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -80.28149 ymin: 26.06761 xmax: -80.27046 ymax: 26.07706
## CRS:            NA
## First 5 geometries:
```

```
## POINT EMPTY
```

```
## POINT (-80.27906 26.06945)
```

```
## POINT EMPTY
## POINT EMPTY
```

```
## POINT (-80.27431 26.06769)
```

An `sftrack` object is simply an `sfc` of `sf_POINTS`, this contrasts with an `sftraj` object which is a mixture of a `GEOMETERYCOLLECTION` and `LINESTRING`. This is because a trajectory can have a start point and an NA end point, a line segment, or an NA and an end point. This allows no-loss conversion back and forth between `sftrack` and an `sftraj`, and because linestrings can not have a NULL point in them.

```
my_sftraj$geometry
```

```
## Geometry set for 445 features
## geometry type:  GEOMETRY
## dimension:      XY
## bbox:           xmin: -80.28149 ymin: 26.06761 xmax: -80.27046 ymax: 26.07706
## CRS:            NA
## First 5 geometries:
```

```
## GEOMETRYCOLLECTION (POINT EMPTY, POINT (-80.279...
```

```
## GEOMETRYCOLLECTION (POINT (-80.27906 26.06945),...
## GEOMETRYCOLLECTION (POINT EMPTY, POINT EMPTY)
## GEOMETRYCOLLECTION (POINT EMPTY, POINT (-80.274...
## LINESTRING (-80.27431 26.06769, -80.2793 26.06867)
```

This does mean that not all `sf` functions will handle an `sftraj` object like it would an `sftrack` if there are NAs in the data set. To help with working with sftraj objects, there are two functions that help extract points from `sftraj` objects.

coord_traj

This function returns a data.frame (x,y,z) of the beginning point of each sftraj geometry.

```
coord_traj(my_sftraj$geometry)[1:10,]
```

```
##           X        Y
## 1        NA       NA
## 1 -80.27906 26.06945
## 1        NA       NA
## 1        NA       NA
##   -80.27431 26.06769
##   -80.27930 26.06867
##   -80.27908 26.06962
## 1 -80.27902 26.06963
## 1        NA       NA
##   -80.27900 26.06982
```

pts_traj

And `pts_traj` returns a list of the beginning point of each sftraj geometry.

```
pts_traj(my_sftraj$geometry)[1:10]
```

```
## [[1]]
## POINT EMPTY
##
## [[2]]
## POINT (-80.27906 26.06945)
##
## [[3]]
## POINT EMPTY
##
## [[4]]
## POINT EMPTY
##
## [[5]]
## POINT (-80.27431 26.06769)
##
## [[6]]
```

```
## POINT (-80.2793 26.06867)

##
## [[7]]

## POINT (-80.27908 26.06962)

##
## [[8]]

## POINT (-80.27902 26.06963)

##
## [[9]]

## POINT EMPTY

##
## [[10]]

## POINT (-80.279 26.06982)
```

is_linestring

May help if you'd like to quickly filter an `sftraj` object to just contain pure linestrings. `is_linestring()` returns TRUE or FALSE if the geometry is a linestring. This does not recalculate anything, it just filters out steps that contained NAs in either phase.

```
is_linestring(my_sftraj$geometry)[1:10]
```

```
##  [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

```
new_sftraj <- my_sftraj[is_linestring(my_sftraj$geometry),]
head(new_sftraj)
```

```
## This is an sftraj object
## crs: NA
## bursts : total = 2 | active burst = id
## Rows: 6 | Cols: 14
##    sensor_code   utc_date utc_time latitude longitude height hdop vdop fix
## 5         CJ11 2019-01-19 04:02:30 26.06769 -80.27431    858  5.1  3.2  2D
## 6         CJ11 2019-01-19 05:02:30 26.06867 -80.27930    350  1.9  3.2  3D
## 7         CJ11 2019-01-19 06:02:30 26.06962 -80.27908     11  2.3  4.5  3D
## 10        CJ11 2019-01-19 17:02:30 26.06982 -80.27900     NA  2.0  3.3  3D
## 11        CJ11 2019-01-19 18:02:05 26.06969 -80.27894      8  4.2  2.5  3D
## 12        CJ11 2019-01-19 19:02:04 26.07174 -80.27890     -3  0.9  1.5  3D
##       acquisition_time                time month                burst
## 5  2019-01-19 04:02:30 2019-01-19 04:02:30     1 (id: CJ11, month: 1)
## 6  2019-01-19 05:02:30 2019-01-19 05:02:30     1 (id: CJ11, month: 1)
## 7  2019-01-19 06:02:30 2019-01-19 06:02:30     1 (id: CJ11, month: 1)
## 10 2019-01-19 17:02:30 2019-01-19 17:02:30     1 (id: CJ11, month: 1)
## 11 2019-01-19 18:02:05 2019-01-19 18:02:05     1 (id: CJ11, month: 1)
## 12 2019-01-19 19:02:04 2019-01-19 19:02:04     1 (id: CJ11, month: 1)
##                           geometry
## 5  LINESTRING (-80.27431 26.06...
## 6  LINESTRING (-80.2793 26.068...
## 7  LINESTRING (-80.27908 26.06...
## 10 LINESTRING (-80.279 26.0698...
## 11 LINESTRING (-80.27894 26.06...
```

```
## 12 LINESTRING (-80.2789 26.071...
```