An Inclusive, Unifying API for Progress Updates

@HenrikBengtsson

University of California San Francisco, R Foundation, R Consortium
Applying a slow function to a vector

```r
slow <- function(x) {
  Sys.sleep(1.0)
  sqrt(x)
}

x <- 1:50
y <- lapply(x, function(z) {
  slow(z)
})

=> 50 seconds to complete
```
With slow functions we want to know ... 

● Is it still running? 
  / Processing ... 

● How much longer? (secs, mins, hours, days) 
  [==========>-------------------]  40% ETA    8s
Progress updates in R
utils::txtProgressBar() - built-in & basic

x <- 1:50
pb <- txtProgressBar(max=length(x))
y <- lapply(x, function(z) {
  setTxtProgressBar(pb, getTxtProgressBar(pb)+1)
  slow(z)
})

|============                    |  42%
progress: beautiful progress bars

```r
x <- 1:50
pb <- progress::progress_bar$new(total=length(x))
y <- lapply(x, function(z) {
  pb$tick()
  slow(z)
})

[=============>-----------------] 42%
```

by Gábor Csárdi
Things we need to be aware of
Let user control progress updates

```r
snail <- function(x, progress = FALSE) {
  if (progress) pb <- progress::progress_bar$new(total=length(x))

  lapply(x, function(z) {
    if (progress) pb$tick()
    slow(z)
  })
}

> x <- 1:50
> y <- snail(x, progress=TRUE)
```
We must be careful with output

```r
snail <- function(x, progress = FALSE) {
  if (progress) pb <- progress::progress_bar$new(total=length(x))

  lapply(x, function(z) {
    if (!progress) message("z=", z)
    if (progress) pb$tick()
    slow(z)
  })
}

> y <- snail(x)                 # with messages
> y <- snail(x, progress=TRUE)  # no messages
```

library(parallel)
cl <- makeCluster(4)
x <- 1:50

pb <- progress::progress_bar$new(total=length(x))
clusterExport(cl, c("slow", "pb"))
y <- parLapply(cl, x, function(z) {
    pb$tick()
    slow(z)
})

=> no progress bar (output from workers is dropped)
We can do better with R’s condition framework
Can be used with for loops, while loops, lapply, purrr, foreach, ...

API for Developers:

```
p <- progressor(along=x)
p()
```

Developer decides:
where in the code progress updates should be signaled

API for Users:

```
with_progress({ expr })
```

User decides:
if, when, and how progress updates are presented
It’s all about signalling progress

```r
snail <- function(x) {
  p <- progressr::progressor(along=x)
  lapply(x, function(z) {
    p()       # signal a progress condition
    slow(z)
  })
}
#
# Just a regular function
> x <- 1:50
> y <- snail(x)
>
```
User decides how progress is presented

```r
# without progress updates
> x <- 1:50
> y <- snail(x)

> with_progress(y <- snail(x))
[======>-----------------] 40%

> handlers("progress", "beepr")
> with_progress(y <- snail(x))
[======>-----------------] 40%
♫ ♪ ♪ ♪ ♪ ♪ ♪ ♪ ♪ ♪ ...
♫
```

beepr by Rasmus Bååth
Worry free use of cat() and message()

```r
snail <- function(x) {
    p <- progressr::.progressor(along=x)
    lapply(x, function(z) {
        message("z=", z)  # No worries!
        p()
        slow(z)
    })
}

> with_progress(y <- snail(1:10))
z=1
z=2
[======>---------------------] 20%
```
Parallelization: progressr + future = ❤

snail <- function(x) {
  p <- progressor(along=x)
  future_lapply(x, function(z) {
    p()
    slow(z)
  })
}

> plan(multisession)
> with_progress(y <- snail(x))
[=============] 40%

Works with any future API:
- future.apply
- furrr
- foreach /w doFuture
- BiocParallel
library(future)
plan(cluster, workers = c("w1.remote.org", "w2.remote.org"))

with_progress({
  y <- snail(1:1000) # parallelization via futures internally
})
[======>-----------]  40%
Take-home messages

- Developer decides *what* progress to report on
  \[ p \leftarrow \text{progressor}(\ldots), \ p() \]

- End-user decide *when and how* progress is reported
  \[ \text{with\_progress}(\ldots), \ \text{handler}(\ldots) \]

- Output doesn’t clash with existing progress information

- Works with parallel processing using futures
Create new progress handlers for end-users

Existing:
- `utils::txtProgressBar()` [default]
- `progress::progress_bar()`
- `beepr::beep()`
- Shiny, ...

I encourage you to build handlers for:
- Pushbullet, Twitter, Telegram, SMS
- Email notifications
- Change color on a smart light bulb (ZigBee)
- ...
There's no limit to what you can do

> handler("ransid_image")
> with_progress(y <- snail(1:1000))

Thank you! 💜

@HenrikBengtsson

HenrikBengtsson/progressr
install.packages("progressr")

ransid by Mike FC