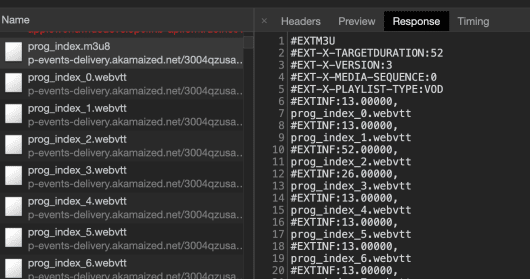
I really don’t care about “memojis” and I have serious dismay over what is a pretty obvious fact that Apple intends to dumb down computing by shifting most folks from Macs to iPads. Apple’s new “Pro” is for design folks and I’m not holding my breath for them to re-embrace the developer/data science communities with better laptops or smaller.

The “meh” hardware/software announcements aren’t the worst parts of these events. The TED-esque scripting (including many failed attempts at faux “authentic” humor) is also becoming quite tedious. I joked about analyzing the “adverbs per minute” but it took a few days for their [WWDC 2019 keynote video](https://developer.apple.com/videos/play/wwdc2019/101/) **with** a subtitle track to emerge. As a result, current time constraints prevent a dive into the subtitles themselves, **but** that doesn’t mean *you* can’t have some fun with them.

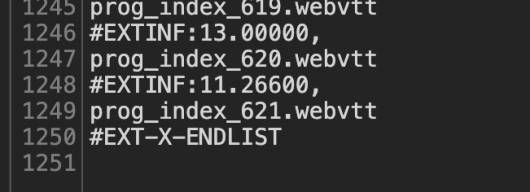
Read on to see how I scraped the subtitles or skip to the end to read more about this “Reader Challenge”.

**Not So Subtle Subtitle Scraping**

If you go to the aforelinked WWDC video URL you’ll see control on the lower right to add a subtitle track. If you do that with browser Developer Tools open you’ll see what that does:

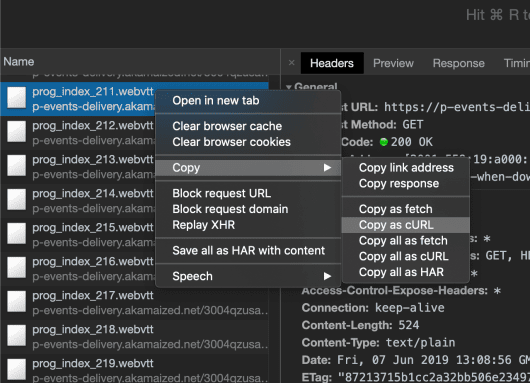
[](https://i0.wp.com/rud.is/b/wp-content/uploads/2019/06/01-subtitle-index.png?ssl=1)

These are [WebVTT](https://www.w3.org/TR/webvtt1/) formatted subtitles which have a format/syntax that enable them to be displayed at the correct playback timecode. We can see how many of them there are by looking at the end of the file:

[](https://i1.wp.com/rud.is/b/wp-content/uploads/2019/06/02-subtitle-count.png?ssl=1)

So, there are 621 of them and each are requested individually (and super-fast, in-parallel). What do these individual requests look like? Just select one of them to take a look. They’re just plain text responses (it’s not a super-intricate format).

Let’s grab one of them to the clipboard and use the {curlconverter} package to turn that into an httr::GET() request via the straighten() and make\_req() functions:

[](https://i0.wp.com/rud.is/b/wp-content/uploads/2019/06/03-subtitle-curl.png?ssl=1)

I went ahead and wrapped it into a fairly-well-named function, but the GET request is virtually untouched from the aforementioned process. I just added the {idx} template into the request URL so we can glue() the right index into it. It is likely that some headers could have been eliminated but I just went with what {curlconverter} processed and returned this time.

library(stringi)

library(subtools)

library(tidytext)

library(purrrogress) # tidy progress bars for free!

library(tidyverse)

library(curlconverter)

#' Fetches a subtitle by index from the 2019 Apple WWDC Keynote subtitle stream

get\_subtitle <- function(idx = 1) {

st\_url <- "https://p-events-delivery.akamaized.net/3004qzusahnbjppuwydgjzsdyzsippar/vod3/cc2/eng4/prog\_index\_{idx}.webvtt"

st\_url <- glue::glue(st\_url)

httr::GET(

url = st\_url,

httr::add\_headers(

`sec-ch-ua` = "Google Chrome 75",

`Sec-Fetch-Mode` = "cors",

Origin = "https://developer.apple.com",

`User-Agent` = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_14\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.80 Safari/537.36",

Referer = "https://developer.apple.com/videos/play/wwdc2019/101/",

`Sec-Fetch-Dest` = "empty",

`Sec-Fetch-Site` = "cross-site"

)

) -> res

out <- httr::content(res, as = "text", encoding = "UTF-8")

out <- stringi::stri\_split\_lines(out)

purrr::flatten\_chr(out)

}

Let’s see what one looks like:

(tmp <- get\_subtitle(1))

## [1] "WEBVTT"

## [2] "X-TIMESTAMP-MAP=MPEGTS:181083,LOCAL:00:00:00.000"

## [3] ""

## [4] "3"

## [5] "00:00:21.199 --> 00:00:22.333"

## [6] ">> FEMALE SPEAKER:"

## [7] "Don't stay up too late."

## [8] ""

## [9] ""

Looking good! But, it’s just plain characters and I don’t feel like writing a subtitle parser. And, I dont’ have to! {subtools} package which we can use. But, it (used to) only work on *files*. It now works on character vectors as well (but you’ll need to install it from my fork until the PR is merged). Let’s turn this set of noise into something we can use:

Library(subtoools)

as\_subtitle(tmp, format = "webvtt") %>%

flatten\_df()

## # A tibble: 1 x 4

## ID Timecode.in Timecode.out Text

##

## 1 3 00:00:21.199 00:00:22.333 >> FEMALE SPEAKER: Don't stay up to…

*So tidy!*

We now need to get all of the subtitles. We’ll do that fast since the video player retrieves them even faster than this iteration does:

# no crawl delay b/c the video player grabs these even faster than this code does

map(1:621, with\_progress(get\_subtitle)) %>% # with\_progress gets you a progress bar for free

map(as\_subtitle, format = "webvtt") %>%

flatten\_df() %>%

as\_tibble() -> apple\_subs

apple\_subs

## # A tibble: 3,220 x 4

## ID Timecode.in Timecode.out Text

##

## 1 3 00:00:21.199 00:00:22.333 >> FEMALE SPEAKER: Don't stay up t…

## 2 4 00:01:10.933 00:01:11.933 >> MALE SPEAKER: Come on.

## 3 5 00:01:36.500 00:01:37.166 >> MALE SPEAKER: All right.

## 4 6 00:01:40.966 00:01:41.733 >> MALE SPEAKER: Yes.

## 5 7 00:01:45.733 00:01:46.666 >> MALE SPEAKER: Woo.

## 6 8 00:01:46.733 00:01:47.833 This is good.

## 7 9 00:01:49.566 00:01:52.666 (Music playing)

## 8 10 00:02:05.200 00:02:12.533 (Applause)

## 9 10 00:02:05.200 00:02:12.533 (Applause)

## 10 11 00:02:14.400 00:02:15.566 >> TIM COOK: Wow.

## # … with 3,210 more rows

Streaming subtitles aren’t error-free and often get duplicated, let’s see if that’s the case:

# Any dups?

distinct(apple\_subs)

## # A tibble: 2,734 x 4

## ID Timecode.in Timecode.out Text

##

## 1 3 00:00:21.199 00:00:22.333 >> FEMALE SPEAKER: Don't stay up t…

## 2 4 00:01:10.933 00:01:11.933 >> MALE SPEAKER: Come on.

## 3 5 00:01:36.500 00:01:37.166 >> MALE SPEAKER: All right.

## 4 6 00:01:40.966 00:01:41.733 >> MALE SPEAKER: Yes.

## 5 7 00:01:45.733 00:01:46.666 >> MALE SPEAKER: Woo.

## 6 8 00:01:46.733 00:01:47.833 This is good.

## 7 9 00:01:49.566 00:01:52.666 (Music playing)

## 8 10 00:02:05.200 00:02:12.533 (Applause)

## 9 11 00:02:14.400 00:02:15.566 >> TIM COOK: Wow.

## 10 12 00:02:15.633 00:02:18.166 Thank you.

## # … with 2,724 more rows

apple\_subs <- distinct(apple\_subs)

There were dups, but not anymore!

You can get that data frame via: <http://rud.is/dl/2019-wwdc-keynote-subtitles.csv.gz>.

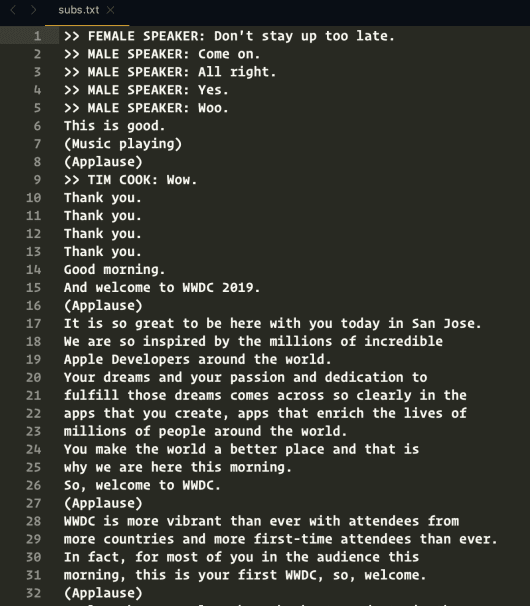
I wanted to see if these looked OK so I dumped just the text to a file and open them up in Sublime Text to spot check:

apple\_subs %>%

pull(Text) %>%

write\_lines("/tmp/subs.txt")

system("subl /tmp/subs.txt") # dblchk

[](https://i1.wp.com/rud.is/b/wp-content/uploads/2019/06/04-subtitle-check.png?ssl=1)

Since we have a good capture of what was spoken, we can start the analysis process:

distinct(apple\_subs) %>%

filter(!grepl("^\\(|^>>", Text)) %>%

unnest\_tokens(word, Text) %>%

anti\_join(get\_stopwords()) %>%

count(word, sort=TRUE)

## Joining, by = "word"

## # A tibble: 2,408 x 2

## word n

##

## 1 now 246

## 2 can 205

## 3 new 142

## 4 like 119

## 5 just 106

## 6 app 77

## 7 great 74

## 8 apple 69

## 9 right 64

## 10 apps 59

## # … with 2,398 more rows

And, that’s when I’ve run out of time.

**Reader Challenge**

You’ve got the cleaned WWDC 2019 Keynote subtitle track and access to my brutal WWDC 2019 Twitter thread. What fun can you have with it? I’d still like to know the adverbs-per-‘n’ (and what kind they were). But, what else can you discover? Is there a pattern of emotional manipulation through word choices at different times? Did they change tone/style throughout the event? What other questions can you ask and tease out with data?

Drop links to your creations (and separate links to code) in the comments and I’ll re-broadcast them on Twitter and gather them all up into a new post to see what y’all came up with.