InfoWorld

Home • Analytics • How to create your own RSS reader with R

by **Sharon Machlis**
Contributing Writer

How to create your own RSS reader with R

How-To
Dec 28, 2022 • 15 mins

JavaScript R Language Web Development

Sure, you could use one of the commercial or open-source RSS readers. But isn’t it more fun to code your own?

RSS feeds have been around since the late ’90s, and they remain a handy way to keep up with multiple news sources. Choose your feeds wisely, and your RSS reader will let you easily scan headlines from multiple sources and stay up to date on fast-moving topics. And while there are several capable commercial and open-source RSS readers available, it’s a lot more satisfying to code your own.

It’s surprisingly easy to create your own RSS feed reader in R. Just follow these eight steps.

Create a Quarto document or R script file

You can use a plain R script, but Quarto adds some useful, out-of-the-box styling. Quarto also gives you easier access to using JavaScript for the final display if you so choose. But the tutorial code works fine in an R file, too.

Unlike an R script, though, my Quarto document needs a YAML header to start. I'll add a few settings in the YAML to generate a single HTML file (embed-resources: true), and not display my code (echo: false) or any code messages or warnings:

```
---
title: "Sharon's RSS Feed"
format:
  html
embed-resources: true
editor: source
execute:
  echo: false
  warning: false
  message: false
---
```

Load needed packages

Next, I'll add some R code inside an R code block (` ``{r} and ``` enclose a block of executable code in Quarto; you don't need those if you're using a plain R script) and load the packages I'll need. As you might guess from its name, tidyRSS is a library for reading RSS feeds into R.

```
``{r}
library(tidyRSS)
library(dplyr)
library(DT)
library(purrr)
library(stringr)
library(lubridate)
```
```

## Add RSS feeds

Selecting relevant feeds is a key part of a useful RSS reader experience. I find mine based on sources I like and then checking websites or
```

searching to see if RSS feeds exist. (As an optional exercise, you can use the **rvest package [https://rvest.tidyverse.org/]** to read sitemaps and wrangle them into RSS format, but that's beyond the scope of this tutorial. Maybe in a future article!)

You may want to store your feeds in a separate CSV or Excel file and have your app import them. This way, you don't have to touch the app code each time you update your feed list. For the sake of demo simplicity here, though, I'll create a data frame in my script file with the feeds I want and my titles for each.

Since I write for both InfoWorld and Computerworld I'll add both of those feeds. In addition, I'll pull in a few R-specific RSS feeds, including R-Bloggers, R Weekly, and Mastodon's #rstats and #QuartoPub RSS feeds at fosstodon.org, the Mastodon instance I use. In the code below, I save the feed info to a data frame call myfeeds with both feed URLs and my desired title for each feed. I then arrange them by feed title:

~~~
```{r}
myfeeds <- data.frame(feed_title = c("All InfoWorld",
                                  "All Computerworld",
                                  "Mastodon rstats",
                                  "Mastodon QuartoPub",
                                  "R Bloggers",
                                  "R Weekly"),
                     feed_url =
c("https://www.infoworld.com/index.rss",

"https://www.computerworld.com/index.rss",

"http://fosstodon.org/tags/rstats.rss",

"http://fosstodon.org/tags/QuartoPub.rss",
                      "https://feeds.feedburner.com/Rbloggers",
                      "https://rweekly.org/atom.xml")
          ) |>
  arrange(feed_title)
```
~~~

Note: From here on, I won't be including the ` ```{r} ``` ` Quarto code "fences" around the R code. All the rest of the R code still needs to be "fenced" in a Quarto doc, though.

Get all the feeds into the same format

This is the most manual part of the process. Ideally, all feeds would be structured exactly the same way, be in the format I want, and never have missing data. In the real world, of course, RSS data can be as messy as any other data set. So, I want to check my feeds and see if/how they need to be cleaned.

In addition, I want to be able to import atom feeds like R-Bloggers' as well as RSS feeds, which means I need to account for those.

To keep things simple, my reader will only display title, item date/time updated, item description, and a way to click to the original (the URL) item.

I'll start by importing each of my feeds into R using tidyRSS, but as a list, one list entry for each feed, and then examine each to see what problems may arise.

```
feed_test <- map(myfeeds$feed_url, tidyfeed)
```

I'm not including that code above in my final RSS reader file; it's for development only.

Create a feed wrangling function

My wrangling function starts simply enough:

```
wrangle_feed <- function(the_feed_url, the_feed_dataframe =
myfeeds) {
  my_feed_data <- tidyRSS::tidyfeed(the_feed_url)
  return(my_feed_data)
}
```

I'd like the feed title to be what I call it in my spreadsheet, not what the feed creator titled it. So, I'll use my feed data frame to look up the title and replace the existing feed title with this code:

```
my_feed_data$feed_title <-
the_feed_dataframe$feed_title[the_feed_dataframe$feed_url ==
the_feed_url][1]
```

I want to select `item_title` , `item_date` , `item_description` , and `item_link` . But if it's an atom feed, those will be called something different: `entry_title` , `entry_last_updated` , `entry_content` , and `entry_url` . Before I select the columns I want, I'll check if it's an atom feed and, if so, rename the atom columns with

```
    if("entry_url" %in% names(my_feed_data)) {
      my_feed_data <- my_feed_data |>
        rename(item_title = entry_title, item_pub_date =
  entry_last_updated, item_link = entry_url, item_description =
  entry_content)
    }
```

Mastodon RSS feeds don't have titles for the posts. I could add the same default title to each post, such as a generic "Mastodon Post," but I'd prefer a title like "Mastodon Post by {username}." Most Mastodon post URLs include the author handle starting with @, although occasionally one won't. I can extract the user name from the mastodon URL and add a custom title with the code below, defaulting to "Mastodon Post" if there is no obvious author in the link.

```
  if(str_detect(my_feed_data$feed_title[1], "Mastodon")) {
    my_feed_data <- my_feed_data |>
      mutate(
        item_author = str_replace_all(item_link, "^.*?/(@.*?)/.*?
  $", "1"),
        item_title = if_else(str_detect(item_author, "@"),
  paste0("Mastodon Post by ", item_author), "Mastodon Post")
      )
    }
```

It's easy for me to find all the Mastodon feeds because I included "Mastodon" in those feed titles.

The `str_replace_all()` code uses a regular expression to find the author in the URL. The pattern `"^.*?/(@.*?)/.*?$"` will drop everything from the start of the string to the / before an @, keep everything from the @ until just before the next /, and then drop everything else.

Next I'll do some additional data wrangling, including selecting and renaming the columns I want and making each item clickable back to the original source.

The code below selects and renames columns and also creates a clickable headline column.

```
 my_feed_data <- my_feed_data |>
  select(Headline = item_title, Date = item_pub_date, URL =
item_link,
         Description = item_description, Feed = feed_title) |>
  mutate(
    Headline = str_glue("<a target='_blank' title='{Headline}'
href='https://www.infoworld.com/{URL}' rel="noopener">{Headline}
</a>")
)
```

Many people like clickable headlines. However, I prefer a clickable >> at the end of the description instead of a clickable headline. The code below is one way to do that.

```
my_feed_data <- my_feed_data |>
  select(Headline = item_title, Date = item_pub_date, URL =
item_link, Description = item_description, Feed = feed_title) |>
  mutate(
    Description = str_glue("{Description},  <a target='_blank'
href='https://www.infoworld.com/{URL}' rel="noopener"> >></a>"),
  )
```

Add some optional data tweaks

The code so far is enough to generate data for a basic feed reader, but the app will look better with some optional tweaks.

For example, the R Bloggers atom feed includes full blog content, but I don't want to download full content into my RSS reader because that makes quick scanning more difficult. Other descriptions may be longer than I'd like as well.

Below is a function that trims the description after `max_chars` number of characters—but at the nearest complete word, so as not to cut off in the middle of a word. It then adds ellipses. The function first checks that there's a description at all, so the code won't break if the description is missing.

```
trim_if_too_long <- function(item_description, max_chars = 600) {
  if(!is.na(item_description)) {
    if(nchar(item_description) > max_chars) {
      item_description <-  stringr::str_sub(item_description, 1,
max_chars)
      item_description <-  str_replace_all(item_description,
"s[^s]+$", ". . . ")
    }
    return(item_description)
  } else {
      return("")
    }
}
```

The function only makes changes if the item description is greater than `max_chars` (currently defaulting to 600). If the description is in fact longer, the first line of code trims the text to `max_chars` length. The second line uses a regular expression to replace anything that's a space followed by one or more characters that aren't spaces at the end of the text string with an ellipsis. In other words, the regex removes any incomplete words at the end of the description and then adds three dots.

If you want to use this function in your RSS reader, make sure to place it *above* the `wrangle_feed` function definition in your Quarto doc or R script.

To apply the function to each feed's description, I'll use purrr's `map_chr()` function:

```
map_chr(Description, trim_if_too_long)
```

and add that to my feed wrangling *before* I add my clickable >> arrows:

8bn2j21i_LxF8qk2n5dQxSSUSpC1OZ92dxdR7AlCWZlL8_tyWdQ4Yjcor9Y9xrcDLSP
U9loOnp92iYTOi9lJLPE4WjFWkus-6wVYiSz8Zlruw-EZYibjKQvtfNYkFygvTpVZ6wq
ShSMpMe6u96JaZk_zcU5TcsQFhb7OIvdFQYkaBCs3fj8nDkQEoX4w04SanRmYR_I
VUHGn90fbu_DN0PgBmzF_uuTDtwi4eccpmyZYY-AyWgJa-v1jI0DZV8VYF6Md-Az4
KnRN7OClJezTzjz7BO_bGGxV4L_8Wug76d4q6UvRekiYPMbLSLRV-2X8I6PBETwfuF
I7X76ifMPqnNtvqQXKuo72FxVrSVQlMWquE67RSOsIab2XRb2CRhi8jsXBfYkQyoLQ
tejdRYp0EyTmK5Kkfg-n39QBKbgM3AhgKfbag9wrKNbrO4-BDb_uQoc5FZLjc27Kc
p9AQd08=&ord=58969581&ntv_ht=6NZJaAA&ntv_tad=16&ntv_enc_pr=vp0uZuWD
WltRkHI24vSHHf9fQodTqsQfzUE9G5eiJ3e2PSbHeedtN2UP7El_fnrqMOygpNhroIN
ZEWX-eMWOluSMY0HSZ4PeYw_RG5p--qE=&ntv_r=https://www.csoonline.com/ar
ticle/3990385/youve-already-been-targeted-why-patch-management-is-mission-
critical.html]

```
 my_feed_data <- my_feed_data |>
  select(Headline = item_title, Date = item_pub_date, URL =
item_link, Description = item_description, Feed = feed_title) |>
  mutate(
      Description = purrr::map_chr(Description,
trim_if_too_long),
      Description = str_glue("{Description},  <a target='_blank'
href='https://www.infoworld.com/{URL}' rel="noopener"> >></a>"),
  )
```

A few of the feeds I've chosen include "To read this article in full, please click here" text at the end, but that's not clickable. It's easy to remove text like that with `str_replace_all()` .

If you want to use this code in your app, make sure to add that code *before* you do any other description wrangling:

```
 my_feed_data <- my_feed_data |>
  select(Headline = item_title, Date = item_pub_date, URL =
item_link, Description = item_description, Feed = feed_title) |>
  mutate(
    Description = str_remove_all(Description, "To read this
article in full, please click here"),
    Description = purrr::map_chr(Description, trim_if_too_long),
    Description = str_glue("{Description},  <a target='_blank'
href='https://www.infoworld.com/{URL}' rel="noopener"> >></a>")
  )
```

One more small nit: I don't like my date/time displaying like 2022-11-16T08:00:00Z. The lubridate package's `format_ISO8601()` function makes it easy to set the desired precision—in this case, I want `ymdhm` but not seconds. After that, I'll replace the "T" with a space so the date column will appear in a format such as 2022-12-21 18:44.

```
Date = format_ISO8601(Date, precision = "ymdhm"),
Date = str_replace_all(Date, "T", " ")
```

Below is my full `wrangle_feed()` function (not showing the separate `trim_if_too_long()` function above it).

```
wrangle_feed <- function(the_feed_url, the_feed_dataframe =
myfeeds) {
  my_feed_data <- tidyfeed(the_feed_url)
  my_feed_data$feed_title <-
the_feed_dataframe$feed_title[the_feed_dataframe$feed_url ==
the_feed_url][1]
 if("entry_url" %in% names(my_feed_data)) {
    my_feed_data <- my_feed_data |>
      rename(item_title = entry_title, item_pub_date =
entry_last_updated, item_link = entry_url, item_description =
entry_content)
 }
 if(str_detect(my_feed_data$feed_title[1], "Mastodon")) {
    my_feed_data <- my_feed_data |>
      mutate(
        item_author = str_replace_all(item_link, "^.*?/(@.*?)/.*?
$", "1"),
        item_title = if_else(str_detect(item_author, "@"),
paste0("Mastodon Post by ", item_author), "Mastodon Post")
      )
 }
 my_feed_data <- my_feed_data |>
  select(Headline = item_title, Date = item_pub_date, URL =
item_link, Description = item_description,  Feed = feed_title) |>
  mutate(
    Description = str_remove_all(Description, "To read this
article in full, please click here"),
    Description = purrr::map_chr(Description, trim_if_too_long),
    Description = str_glue("{Description},  <a target='_blank'
href='https://www.infoworld.com/{URL}' rel="noopener"> >></a>"),
    Date = format_ISO8601(Date, precision = "ymdhm"),
    Date = str_replace_all(Date, "T", " ")
    )
return(my_feed_data)
}
```

Handle a missing or broken feed

I want to make sure this code doesn't blow up and stop on a single error if one of the feeds is unavailable. I can do that by making a "safe," error-handling version of the function with purrr's `possibly()` :

```
wrangle_feed_safely <- possibly(wrangle_feed, otherwise = NULL)
```

The `wrangle_feed_safely()` version of the function returns NULL if there's an error instead of stopping. Now I can run the function on all my feed URLs and get a single data frame returned with purrr's `map_df()` . The code below also arranges results by descending date so the newest entries appear first, regardless of source.

```
mydata <- map_df(myfeeds$feed_url, wrangle_feed_safely) |>
  arrange(desc(Date))
```

Display the results

The hard part is done, we have our data! Now it's time to display the results.

I'll make a copy of the data frame without the URL field for use in my display table, since I don't want to show the URL field (I've got the clickable >> in my description). I wouldn't make a copy for a huge data set, but this is small, and it's a bit of a backup in case I decide later on that I still want the URL field.

```
mytabledata <- select(mydata, -URL)
```

One of the easiest ways to display this data is with a table. I'll use the DT package because I like its ability to use regular-expression searching. Regex searching is especially handy when searching for something like "R," because a regex lets you search for patterns such as R as a separate word and not just R that might be starting any capitalized word.

In my Quarto document, I'll enclose the table code chunk in a "column-page" CSS style class with `:::{.column-page}` at the top and `:::` at the end, as you can see in the code below. That tells my Quarto document to make the table wider than usual—a full page width. column-page is a built-in CSS style that increases the content width. But you don't have to know how to code HTML and CSS in order to make this modification.

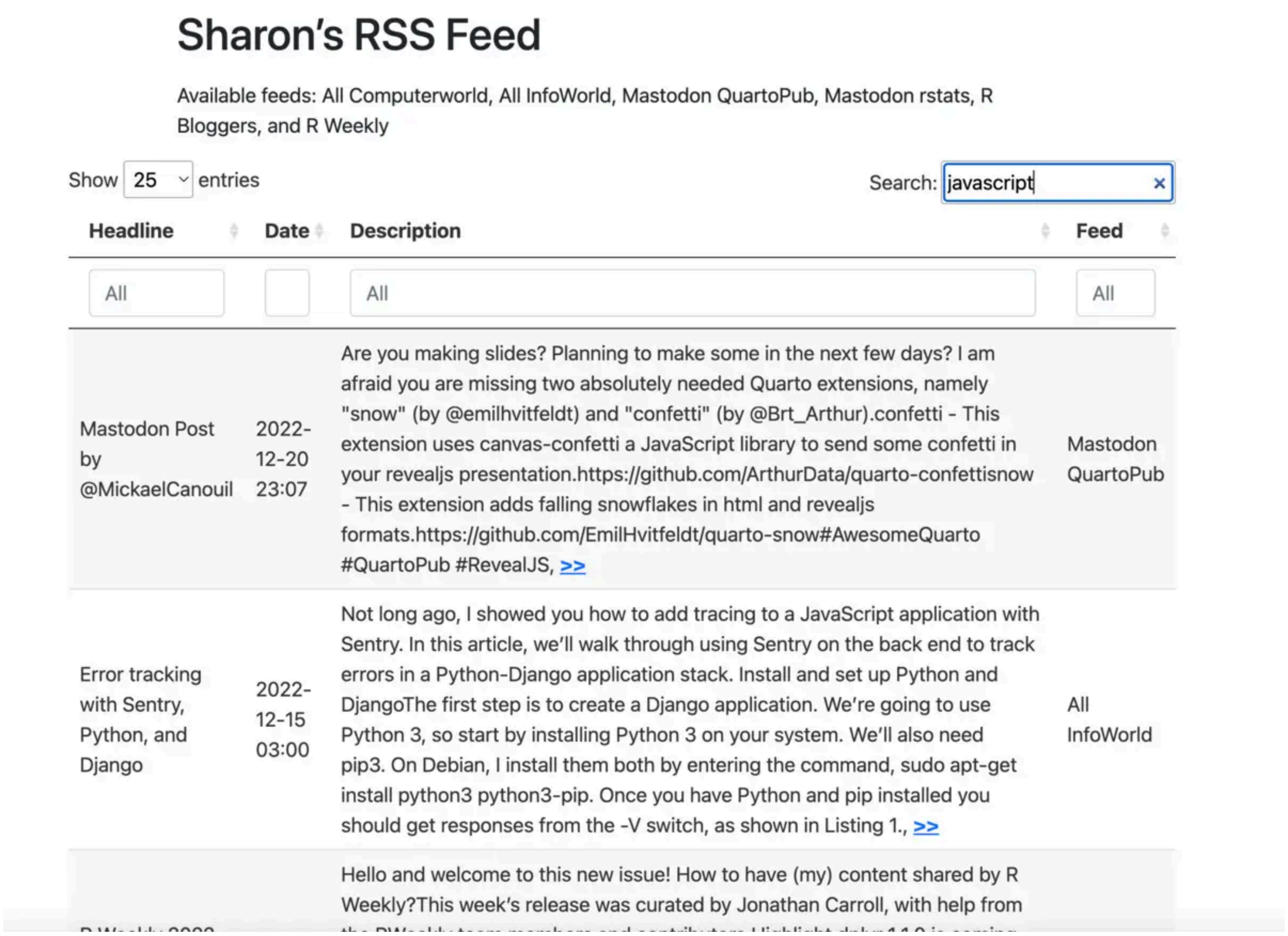
If this option still isn't wide enough (sometimes the table still scrolls because of, say, a ridiculously long URL in a post that won't line break), you can use `{.column-screen}` instead of `{.column-page}` to remove the page margins altogether.

The code below also makes some tweaks to the default DT datatable. `filter = 'top'` adds search filters above each column. `escape = FALSE` displays HTML as HTML instead of showing the underlying code. I add `regex=TRUE` and `caseInsensitive=TRUE` and ignore-case searching to the search options. I also tweak the page length and page length menu options, and set my third column (Description) to be 80% of the table width. (If you're wondering why the target column is `2` when I want the *third* column, it's because DT is a wrapper for a JavaScript library, and the underlying library uses the JS convention of starting to count at 0).

~~~
:::{.column-page}

```{r}
DT::datatable(mytabledata, filter = 'top', escape = FALSE,
rownames = FALSE,
  options = list(
  search = list(regex = TRUE, caseInsensitive = TRUE),
  pageLength = 25,
  lengthMenu = c(25, 50, 100, 200),
  autowidth = TRUE,
  columnDefs = list(list(width = '80%', targets = list(2)))
  )
)
```

:::
~~~



Sharon's RSS Feed

Available feeds: All Computerworld, All InfoWorld, Mastodon QuartoPub, Mastodon rstats, R Bloggers, and R Weekly

Show 25 entries Search: javascript

Headline	Date	Description	Feed
All		All	All
Mastodon Post by @MickaelCanouil	2022-12-20 23:07	Are you making slides? Planning to make some in the next few days? I am afraid you are missing two absolutely needed Quarto extensions, namely "snow" (by @emilhvitfeldt) and "confetti" (by @Brt_Arthur).confetti - This extension uses canvas-confetti a JavaScript library to send some confetti in your revealjs presentation.https://github.com/ArthurData/quarto-confettisnow - This extension adds falling snowflakes in html and revealjs formats.https://github.com/EmilHvitfeldt/quarto-snow#AwesomeQuarto #QuartoPub #RevealJS, >>	Mastodon QuartoPub
Error tracking with Sentry, Python, and Django	2022-12-15 03:00	Not long ago, I showed you how to add tracing to a JavaScript application with Sentry. In this article, we'll walk through using Sentry on the back end to track errors in a Python-Django application stack. Install and set up Python and DjangoThe first step is to create a Django application. We're going to use Python 3, so start by installing Python 3 on your system. We'll also need pip3. On Debian, I install them both by entering the command, sudo apt-get install python3 python3-pip. Once you have Python and pip installed you should get responses from the -V switch, as shown in Listing 1., >>	All InfoWorld
		Hello and welcome to this new issue! How to have (my) content shared by R Weekly?This week's release was curated by Jonathan Carroll, with help from	

[https://images.idgesg.net/images/article/2022/12/rssfeed-100935729-orig.jpg?auto=webp&quality=85,70]

Example of the RSS feed reader table, searching for JavaScript entries.

CREDIT: SHARON MACHLIS

Thanks to regex searching, you can search for R as a separate word with the regular expression `bRb` . The `b` indicates a "word boundary" such as a space, punctuation mark, or beginning or end of a line.

And there you have it, a simple RSS reader! There are more modications you could make, including caching results and further tweaking the display. For example, adding

```
Available feeds: `r
knitr::combine_words(sort(unique(mydata$Feed)))`
```

to the Quarto document after parsing the RSS feeds will show a list of all the available feeds.

For more on Quarto and how you might use JavaScript with R in a Quarto document, see "**A beginner's guide to using Observable JavaScript, R, and Python with Quarto** **[https://www.infoworld.com/article/2336848/a-beginners-guide-to-using-observable-javascript-r-and-python-with-quarto.html]**." And for more R tips, head to InfoWorld's **Do More With R page** **[https://www.infoworld.com/article/2262286/do-more-with-r-video-tutorials.html]**.

Sponsored Links

Empower your cybersecurity team with expert insights from Palo Alto Networks.

Secure AI by Design: Unleash the power of AI and keep applications, usage and data secure.