

# xts FAQ

xts Development Team\*†

January 14, 2020

## Contents

|  |   |
|--|---|
| What is <b>xts</b> ? . . . . .   | 2 |
| Why should I use <b>xts</b> rather than <b>zoo</b> or another time-series package? . . . . .   | 2 |
| How do I install <b>xts</b> ? . . . . .  | 2 |
| I have multiple .csv time-series files that I need to load in a single <b>xts</b> object. What is the most efficient way to import the files? . . . . .  | 2 |
| Why is <b>xts</b> implemented as a matrix rather than a data frame? . . . . .  | 2 |
| How can I simplify the syntax when referring to <b>xts</b> object column names? . . . . .  | 2 |
| How can I replace the zeros in an <b>xts</b> object with the last non-zero value in the series? . . . . .  | 3 |
| How do I create an <b>xts</b> index with millisecond precision? . . . . .  | 3 |
| I have a millisecond-resolution index, but the milliseconds aren't displayed. What went wrong? . . . . .   | 3 |
| I set <code>digits.sec=3</code> , but <b>R</b> doesn't show the values correctly. . . . .  | 4 |
| I am using <code>apply</code> to run a custom function on my <b>xts</b> object. Why does the returned matrix have different dimensions than the original one? . . . . .  | 4 |
| I have an <b>xts</b> object with varying numbers of observations per day (e.g., one day might contain 10 observations, while another day contains 20 observations). How can I apply a function to all observations for each day? . . . . .   | 4 |
| How can I process daily data for a specific time subset? . . . . .   | 5 |
| How can I analyze my irregular data in regular blocks, adding observations for each regular block if one doesn't exist in the original time-series object? . . . . .   | 5 |
| Why do I get a <b>zoo</b> object when I call <code>transform</code> on my <b>xts</b> object? . . . . .   | 6 |
| Why can't I use the <code>&amp;</code> operator in <b>xts</b> objects when querying dates? . . . . .   | 6 |
| How do I subset an <b>xts</b> object to only include weekdays (excluding Saturday and Sundays)? . . . . .  | 6 |
| I need to quickly convert a data.frame that contains the time-stamps in one of the columns. Using <code>as.xts(Data)</code> returns an error. How do I build my <b>xts</b> object? . . . . .   | 7 |
| I have two time-series with different frequency. I want to combine the data into a single <b>xts</b> object, but the times are not exactly aligned. I want to have one row in the result for each ten minute period, with the time index showing the beginning of the time period. . . . . | 7 |
| Why do I get a warning when running the code below? . . . . .  | 7 |

---

\*Contact author: Joshua M. Ulrich <josh.m.ulrich@gmail.com>

†Thanks to Alberto Giannetti and Michael R. Weylandt for their many contributions.

## What is xts?

`xts` is an **R** package offering a number of functionalities to work on time-indexed data. `xts` extends `zoo`, another popular package for time-series analysis.

## Why should I use xts rather than zoo or another time-series package?

The main benefit of `xts` is its seamless compatibility with other packages using different time-series classes (`timeSeries`, `zoo`, ...). In addition, `xts` allows the user to add custom attributes to any object. See the main `xts` vignette for more information.

## How do I install xts?

`xts` depends on `zoo` and suggests some other packages. You should be able to install `xts` and all the other required components by simply calling `install.packages('pkg')` from the **R** prompt.

## I have multiple .csv time-series files that I need to load in a single xts object. What is the most efficient way to import the files?

If the files have the same format, load them with `read.zoo` and then call `rbind` to join the series together; finally, call `as.xts` on the result. Using a combination of `lapply` and `do.call` can accomplish this with very little code:

```
> filenames <- c("a.csv", "b.csv", "c.csv")
> sample.xts <- as.xts(do.call("rbind", lapply(filenames, read.zoo)))
```

## Why is xts implemented as a matrix rather than a data frame?

`xts` uses a matrix rather than `data.frame` because:

1. `xts` is a subclass of `zoo`, and that's how `zoo` objects are structured; and
2. matrix objects have much better performance than `data.frames`.

## How can I simplify the syntax when referring to xts object column names?

`with` allows you to use the column names while avoiding the full square brackets syntax. For example:

```
> lm(sample.xts[, "Res"] ~ sample.xts[, "ThisVar"] + sample.xts[, "ThatVar"])
```

can be converted to

```
> with(sample.xts, lm(Res ~ ThisVar + ThatVar))
```

## How can I replace the zeros in an xts object with the last non-zero value in the series?

Convert the zeros to NA and then use `na.locf`:

```
> sample.xts <- xts(c(1:3, 0, 0, 0), as.POSIXct("1970-01-01")+0:5)
> sample.xts[sample.xts==0] <- NA
> cbind(orig=sample.xts, locf=na.locf(sample.xts))
```

|                     | orig | locf |
|---------------------|------|------|
| 1970-01-01 00:00:00 | 1    | 1    |
| 1970-01-01 00:00:01 | 2    | 2    |
| 1970-01-01 00:00:02 | 3    | 3    |
| 1970-01-01 00:00:03 | NA   | 3    |
| 1970-01-01 00:00:04 | NA   | 3    |
| 1970-01-01 00:00:05 | NA   | 3    |

## How do I create an xts index with millisecond precision?

Milliseconds in `xts` indexes are stored as decimal values. This example builds an index spaced by 100 milliseconds, starting at the current system time:

```
> data(sample_matrix)
> sample.xts <- xts(1:10, seq(as.POSIXct("1970-01-01"), by=0.1, length=10))
```

## I have a millisecond-resolution index, but the milliseconds aren't displayed. What went wrong?

Set the `digits.secs` option to some sub-second precision. Continuing from the previous example, if you are interested in milliseconds:

```
> options(digits.secs=3)
> head(sample.xts)
```

|                       | [,1] |
|-----------------------|------|
| 1970-01-01 00:00:00.0 | 1    |
| 1970-01-01 00:00:00.1 | 2    |
| 1970-01-01 00:00:00.2 | 3    |
| 1970-01-01 00:00:00.3 | 4    |
| 1970-01-01 00:00:00.4 | 5    |
| 1970-01-01 00:00:00.5 | 6    |

## I set `digits.sec=3`, but R doesn't show the values correctly.

Sub-second values are stored with approximately microsecond precision. Setting the precision to only 3 decimal hides the full index value in microseconds and might be tricky to interpret depending how the machine rounds the millisecond (3rd) digit. Set the `digits.secs` option to a value higher than 3 or convert the date-time to numeric and use `print`'s `digits` argument, or `sprintf` to display the full value. For example:

```
> dt <- as.POSIXct("2012-03-20 09:02:50.001")
> print(as.numeric(dt), digits=20)
```

```
[1] 1332234170.0009999275
```

```
> sprintf("%20.10f", dt)
```

```
[1] "1332234170.0009999275"
```

## I am using `apply` to run a custom function on my `xts` object. Why does the returned matrix have different dimensions than the original one?

When working on rows, `apply` returns a transposed version of the original matrix. Simply call `t` on the returned matrix to restore the original dimensions:

```
> sample.xts.2 <- xts(t(apply(sample.xts, 1, myfun)), index(sample.xts))
```

## I have an `xts` object with varying numbers of observations per day (e.g., one day might contain 10 observations, while another day contains 20 observations). How can I apply a function to all observations for each day?

You can use `apply.daily`, or `period.apply` more generally:

```
> sample.xts <- xts(1:50, seq(as.POSIXct("1970-01-01"),
+ as.POSIXct("1970-01-03")-1, length=50))
> apply.daily(sample.xts, mean)
```

```
          [,1]
1970-01-01 23:30:36.244  13
1970-01-02 23:59:59.000  38
```

```
> period.apply(sample.xts, endpoints(sample.xts, "days"), mean)
```

```
          [,1]
1970-01-01 23:30:36.244  13
1970-01-02 23:59:59.000  38
```

```
> period.apply(sample.xts, endpoints(sample.xts, "hours", 6), mean)
```

```

                                [,1]
1970-01-01 05:52:39.061  4.0
1970-01-01 11:45:18.122 10.5
1970-01-01 17:37:57.183 16.5
1970-01-01 23:30:36.244 22.5
1970-01-02 05:23:15.306 28.5
1970-01-02 11:15:54.367 34.5
1970-01-02 17:08:33.428 40.5
1970-01-02 23:59:59.000 47.0

```

## How can I process daily data for a specific time subset?

First use time-of-day subsetting to extract the time range you want to work on (note the leading "T" and leading zeros are required for each time in the range: "T06:00"), then use `apply.daily` to apply your function to the subset:

```
> apply.daily(sample.xts['T06:00/T17:00'], mean)
```

## How can I analyze my irregular data in regular blocks, adding observations for each regular block if one doesn't exist in the original time-series object?

Use `align.time` to round-up the indexes to the periods you are interested in, then call `period.apply` to apply your function. Finally, merge the result with an empty xts object that contains all the regular index values you want:

```

> sample.xts <- xts(1:6, as.POSIXct(c("2009-09-22 07:43:30",
+   "2009-10-01 03:50:30", "2009-10-01 08:45:00", "2009-10-01 09:48:15",
+   "2009-11-11 10:30:30", "2009-11-11 11:12:45")))
> # align index into regular (e.g. 3-hour) blocks
> aligned.xts <- align.time(sample.xts, n=60*60*3)
> # apply your function to each block
> count <- period.apply(aligned.xts, endpoints(aligned.xts, "hours", 3), length)
> # create an empty xts object with the desired regular index
> empty.xts <- xts(, seq(start(aligned.xts), end(aligned.xts), by="3 hours"))
> # merge the counts with the empty object
> head(out1 <- merge(empty.xts, count))

```

```

                                count
2009-09-22 09:00:00      1
2009-09-22 12:00:00    NA
2009-09-22 15:00:00    NA
2009-09-22 18:00:00    NA
2009-09-22 21:00:00    NA
2009-09-23 00:00:00    NA

```

```

> # or fill with zeros
> head(out2 <- merge(empty.xts, count, fill=0))

```

|                     | count |
|---------------------|-------|
| 2009-09-22 09:00:00 | 1     |
| 2009-09-22 12:00:00 | 0     |
| 2009-09-22 15:00:00 | 0     |
| 2009-09-22 18:00:00 | 0     |
| 2009-09-22 21:00:00 | 0     |
| 2009-09-23 00:00:00 | 0     |

## Why do I get a zoo object when I call transform on my xts object?

There's no `xts` method for `transform`, so the `zoo` method is dispatched. The `zoo` method explicitly creates a new `zoo` object. To convert the transformed object back to an `xts` object wrap the `transform` call in `as.xts`:

```
> sample.xts <- as.xts(transform(sample.xts, ABC=1))
```

You might also have to reset the index timezone:

```
> tzone(sample.xts) <- Sys.getenv("TZ")
```

## Why can't I use the & operator in xts objects when querying dates?

"2011-09-21" is not a logical vector and cannot be coerced to a logical vector. See `?"&"` for details.

`xts`' ISO-8601 style subsetting is nice, but there's nothing we can do to change the behavior of `.Primitive("&")`. You can do something like this though:

```
> sample.xts[sample.xts$Symbol == "AAPL" & index(sample.xts) == as.POSIXct("2011-09-21"),]
```

or:

```
> sample.xts[sample.xts$Symbol == "AAPL"]['2011-09-21']
```

## How do I subset an xts object to only include weekdays (excluding Saturday and Sundays)?

Use `.indexwday` to only include Mon-Fri days:

```
> data(sample_matrix)
> sample.xts <- as.xts(sample_matrix)
> wday.xts <- sample.xts[.indexwday(sample.xts) %in% 1:5]
> head(wday.xts)
```

|            | Open     | High     | Low      | Close    |
|------------|----------|----------|----------|----------|
| 2007-01-02 | 50.03978 | 50.11778 | 49.95041 | 50.11778 |
| 2007-01-03 | 50.23050 | 50.42188 | 50.23050 | 50.39767 |

```
2007-01-04 50.42096 50.42096 50.26414 50.33236
2007-01-05 50.37347 50.37347 50.22103 50.33459
2007-01-08 50.03555 50.10363 49.96971 49.98806
2007-01-09 49.99489 49.99489 49.80454 49.91333
```

**I need to quickly convert a data.frame that contains the time-stamps in one of the columns. Using `as.xts(Data)` returns an error. How do I build my xts object?**

The `as.xts` function assumes the date-time index is contained in the `rownames` of the object to be converted. If this is not the case, you need to use the `xts` constructor, which requires two arguments: a vector or a matrix carrying data and a vector of type `Date`, `POSIXct`, `chron`, ..., supplying the time index information. If you are certain the time-stamps are in a specific column, you can use:

```
> Data <- data.frame(timestamp=as.Date("1970-01-01"), obs=21)
> sample.xts <- xts(Data[,-1], order.by=Data[,1])
```

If you aren't certain, you need to explicitly reference the column name that contains the time-stamps:

```
> Data <- data.frame(obs=21, timestamp=as.Date("1970-01-01"))
> sample.xts <- xts(Data[,!grepl("timestamp",colnames(Data))],
+   order.by=Data$timestamp)
```

**I have two time-series with different frequency. I want to combine the data into a single xts object, but the times are not exactly aligned. I want to have one row in the result for each ten minute period, with the time index showing the beginning of the time period.**

`align.time` creates evenly spaced time-series from a set of indexes, `merge` ensure two time-series are combined in a single `xts` object with all original columns and indexes preserved. The new object has one entry for each timestamp from both series and missing values are replaced with `NA`.

```
> x1 <- align.time(xts(Data1$obs, Data1$timestamp), n=600)
> x2 <- align.time(xts(Data2$obs, Data2$timestamp), n=600)
> merge(x1, x2)
```

**Why do I get a warning when running the code below?**

```
> data(sample_matrix)
> sample.xts <- as.xts(sample_matrix)
> sample.xts["2007-01"]$Close <- sample.xts["2007-01"]$Close + 1
```

```
> #Warning message:
> #In NextMethod(.Generic) :
> # number of items to replace is not a multiple of replacement length
```

This code creates two calls to the subset-replacement function `xts::`[<- .xts``. The first call replaces the value of `Close` in a temporary copy of the first row of the object on the left-hand-side of the assignment, which works fine. The second call tries to replace the first *element* of the object on the left-hand-side of the assignment with the modified temporary copy of the first row. This is the problem.

For the command to work, there needs to be a comma in the first subset call on the left-hand-side:

```
> sample.xts["2007-01",]$Close <- sample.xts["2007-01"]$Close + 1
```

This isn't encouraged, because the code isn't clear. Simply remember to subset by column first, then row, if you insist on making two calls to the subset-replacement function. A cleaner and faster solution is below. It's only one function call and it avoids the `$` function (which is marginally slower on `xts` objects).

```
> sample.xts["2007-01", "Close"] <- sample.xts["2007-01", "Close"] + 1
```