
# Downloading Timeseries from finance.yahoo.com using R

Yvan Lengwiler
Faculty of Business and Economics
University of Basel

yvan.lengwiler@unibas.ch

File: yahoo.tex

**Contents**

## 1 What is `finance.yahoo.com`?

`finance.yahoo.com` is a website that provides easy and free access to a wealth of financial market data. They provide up to date data, as well as historical time series. This guide explains how to download historical time series.

## 2 Equity prices, indexes, yields ...

### 2.1 *"Manually" in the browser*

Browse to `finance.yahoo.com`, enter the symbol of the asset you are interested in, and click on `Historical data`. You can then select the time range you are interested i, and whether you want the data in daily, wekkly, or monthly interval. Click `Apply` and then `download`. This should send a `csv` file to your computer. A `csv` file is a table of data where the columns are separated with commas and rows with new lines. This format is easy to read into various programs (Excel, R Python, etc.)

### 2.2 *The programmatic way*

*a) Constructing the URL*

The key for this is to understand the API, which is a structured, but otherwise simple url:

```
https://query1.finance.yahoo.com/v7/finance/download/SYMBOL
?period1=BEGINDATE&period2=ENDDATE&interval=INTERVAL
&events=history&includeAdjustedClose=true
```

This is all supposed to be one line. The bold parts are variables that you need to set.

**`SYMBOL`** This is the symbol of the asset you wish to download data for. Examples: `GOOG` is Google, `AMD` is Advanced Micro Devices, `^GSPC` is the S&P500 index,`^TNX` is the 10 year US Treasury yield, `CHFEUR=X` is the CHF-EUR exchange rate, etc.

**`BEGINDATE` and `ENDDATE`** These are codes that identify the period for which you require data. You can not just type in a date; rather, the dates are coded in terms of seconds since January 1st, 1970. See the table at the end for more on that below.

**`INTERVAL`** This is either `1d` for daily, `1wk` for weekly, or `1mo` for monthly data.

Once you have constructed your url, you can use it to download the csv file. You can do this directly in the shell, or you can use some programming language to automate this process more and have the data ready for analysis.

*b*) *shell*

In *Windows*, a shell is the environment that you find yourself in when you run the `cmd` command. This is commonly known as a 'command prompt'. In *Linux* you can use any standard shell (e.g. `bash`), on a *Mac* you access the shell when you start the Terminal). You can download data from yahoo in the shell with this command:

**Download data** command line

```
curl --output FILENAME.csv URL
```

The data are stored in `FILENAME.csv` in your current directory. That leaves us with a data file that we can open with Excel and look at.

*c*) *R*

A more elegant way to handle this is to download it directly into a variable of a programming language we use for analysis. This is not overly difficult,

**Download data** R language

```
symbol <- 'AMD'
fromcode <- '1577836800'
tocode <- '1708473600'
interval <- '1mo'

url <- paste0('https://query1.finance.yahoo.com/v7/finance/download/',
symbol, "?period1=", fromcode, "&period2=", tocode,
"&interval=", interval, "&events=history&includeAdjustedClose=true")

data <- read.csv(url, header=TRUE, row.names='Date')
```

## 2.3 *Date codes*

Index 0 ist 1970-01-01. The index counts seconds (!), so one day is $60 * 60 * 24 = 86400$.[1] Because not all months and all years are equally long, it is a pain to compute this if you do not have a date computation package. In that case, you can use the table below.

Of course, it is far more elegant and convenient to compute these numbers programmatically. The next box demonstrates how we can do this.

[1] This is the standard definition of time codes in Unix-like systems.

| date | datecode | date | datecode | date | datecode |
|---|---|---|---|---|---|
| 1970-01-01 | 0 | 1990-01-01 | 631,152,000 | 2010-01-01 | 1,262,304,000 |
| 1971-01-01 | 31,536,000 | 1991-01-01 | 662,688,000 | 2011-01-01 | 1,293,840,000 |
| 1972-01-01 | 63,072,000 | 1992-01-01 | 694,224,000 | 2012-01-01 | 1,325,376,000 |
| 1973-01-01 | 94,694,400 | 1993-01-01 | 725,846,400 | 2013-01-01 | 1,356,998,400 |
| 1974-01-01 | 126,230,400 | 1994-01-01 | 757,382,400 | 2014-01-01 | 1,388,534,400 |
| 1975-01-01 | 157,766,400 | 1995-01-01 | 788,918,400 | 2015-01-01 | 1,420,070,400 |
| 1976-01-01 | 189,302,400 | 1996-01-01 | 820,454,400 | 2016-01-01 | 1,451,606,400 |
| 1977-01-01 | 220,924,800 | 1997-01-01 | 852,076,800 | 2017-01-01 | 1,483,228,800 |
| 1978-01-01 | 252,460,800 | 1998-01-01 | 883,612,800 | 2018-01-01 | 1,514,764,800 |
| 1979-01-01 | 283,996,800 | 1999-01-01 | 915,148,800 | 2019-01-01 | 1,546,300,800 |
| 1980-01-01 | 315,532,800 | 2000-01-01 | 946,684,800 | 2020-01-01 | 1,577,836,800 |
| 1981-01-01 | 347,155,200 | 2001-01-01 | 978,307,200 | 2021-01-01 | 1,609,459,200 |
| 1982-01-01 | 378,691,200 | 2002-01-01 | 1,009,843,200 | 2022-01-01 | 1,640,995,200 |
| 1983-01-01 | 410,227,200 | 2003-01-01 | 1,041,379,200 | 2023-01-01 | 1,672,531,200 |
| 1984-01-01 | 441,763,200 | 2004-01-01 | 1,072,915,200 | 2024-01-01 | 1,704,067,200 |
| 1985-01-01 | 473,385,600 | 2005-01-01 | 1,104,537,600 | 2025-01-01 | 1,735,689,600 |
| 1986-01-01 | 504,921,600 | 2006-01-01 | 1,136,073,600 | 2026-01-01 | 1,767,225,600 |
| 1987-01-01 | 536,457,600 | 2007-01-01 | 1,167,609,600 | 2027-01-01 | 1,798,761,600 |
| 1988-01-01 | 567,993,600 | 2008-01-01 | 1,199,145,600 | 2028-01-01 | 1,830,297,600 |
| 1989-01-01 | 599,616,000 | 2009-01-01 | 1,230,768,000 | 2029-01-01 | 1,861,920,000 |

**yahoo time codes** R language

```
basedate <- as.Date("1970-01-01")
fromdate <- as.Date("2010-01-01")
todate <- as.Date("2023-12-31")
fromcode <- difftime(fromdate, basedate, units="secs")
tocode <- difftime(todate, basedate, units="secs")
```

This is implemented in the script `'Chart-an-Equity.r'` that is available at `https://github.com/yvan-lengwiler/Finance-Uni`.