tsdb: A terribly-simple data base for time series

Enrico Schumann

February 6, 2017

1 About tsdb

A terribly-simple data base for time series. All series are saved as csv files. The package offers utilities for saving files in a standardised format, and for retrieving and joining data.

1.1 Good things about tsdb

- no setup needed, no system dependencies (i.e. external software, such as a database)
- completely portable; moving from one computer to another requires no effort (the only file to take care of is file encoding)
- data usable by other software

1.2 When you need another database

- tsdb is potentially slow
- no multi-user support; no access-rights management (other than that provided by the OS)
- no network protocols

2 Using tsdb

2.1 Writing data

We first load the package.

```
library("tsdb")
```

Start by creating time-series data.

```
library("zoo")
z <- zoo(1:5, as.Date("2016-1-1")+0:4)
z
```

```
2016-01-01 2016-01-02 2016-01-03 2016-01-04 2016-01-05 
         1          2          3          4          5 
```

To store these data, we need to enforce a certain format, which the functions `ts_table` and `as.ts_table` do.

```
ts <- as.ts_table(z, columns = "A")
ts
```

```
5 rows [2016-01-01 -> 2016-01-05]: A
```

Note that we had to provide a column name (`A`) for the data. That is not optional. It is one of the things that `ts_table` enforces. Another is that timestamps need to be of class `Date` or `POSIXct`.

To store the data to a file, use `write_ts_table`. The function will take a directory and file name as arguments, which mimics the hierarchy of databases and tables in a classical database.

```
write_ts_table(ts, dir = "~/tsdb/daily", file = "example1")
```

The written file will look like this:

```
"timestamp","A"
16801,1
16802,2
16803,3
16804,4
16805,5
```

You may notice that the dates been replaced by numbers. The mapping between these numbers and calendar times is described later, when we discuss the representation of timestamps.

Let us write a second file. This time, we use `ts_table` directly.

```
x <- array(1:20, dim = c(10,2))
colnames(x) <- c("A", "B")
x
```

```
       A  B
 [1,]  1 11
 [2,]  2 12
 [3,]  3 13
 [4,]  4 14
 [5,]  5 15
 [6,]  6 16
 [7,]  7 17
 [8,]  8 18
 [9,]  9 19
[10,] 10 20
```

```
ts_table(x, timestamp = as.Date("2016-1-1")+0:9)
```

```
10 rows [2016-01-01 -> 2016-01-10]: A, B
```

We can also explicitly specify the column names, which will override the column names of the data. In fact, this is the preferred way, since it makes things more explicit (which usually means safer).

```
ts <- ts_table(x, timestamp = as.Date("2016-1-1")+0:9,
               columns = c("B", "A"))
ts
```

```
10 rows [2016-01-01 -> 2016-01-10]: B, A
```

We write the data to a file `example2`.

```
write_ts_table(ts, dir = "~/tsdb/daily", file = "example2")
```

The written file looks like this:

```
"timestamp","B","A"
16801,1,11
16802,2,12
16803,3,13
16804,4,14
16805,5,15
16806,6,16
16807,7,17
16808,8,18
16809,9,19
16810,10,20
```

2.2 Reading data

Use the function `read_ts_tables`.

```
read_ts_tables("example1", dir = "~/tsdb/daily", columns = "A")
```

The default return value is a list with components `data`, `timestamp`, `columns` and `file.path`.

```
$data
     [,1]
[1,]    1
[2,]    4
[3,]    5

$timestamp
[1] "2016-01-01" "2016-01-04" "2016-01-05"

$columns
[1] "A"

$file.path
[1] "~/tsdb/daily/example1::A"
```

More convenient may be to specify a `return.class`.

```
read_ts_tables("example1", dir = "~/tsdb/daily", columns = "A",
                  return.class = "zoo")
```

```
           ~/tsdb/daily/example1::A
2016-01-01                        1
2016-01-04                        4
2016-01-05                        5
```

```
read_ts_tables("example1", dir = "~/tsdb/daily", columns = "A",
                  return.class = "data.frame")
```

```
   timestamp ~/tsdb/daily/example1::A
1 2016-01-01                        1
2 2016-01-04                        4
3 2016-01-05                        5
```

But wait. We provided and wrote to the file values for 1 January to 5 January. But we only got values for 1, 4 and 5 January. The reason is that tsdb was written with financial data in mind, and on weekends there are no prices.

```
weekdays(as.Date("2016-1-1")+0:4)
```

```
[1] "Friday"    "Saturday" "Sunday"    "Monday"    "Tuesday"
```

To obtain data for weekends as well, specify the argument `drop.weekends`.

```
read_ts_tables("example1", dir = "~/tsdb/daily",
               columns = "A",
               return.class = "data.frame",
               drop.weekends = FALSE)
```

```
   timestamp ~/tsdb/daily/example1::A
1 2016-01-01                        1
2 2016-01-02                        2
3 2016-01-03                        3
4 2016-01-04                        4
5 2016-01-05                        5
```

You may have noticed a small difference in the names of the functions for reading and writing. We always write a single table, but we read tables.

```
read_ts_tables(c("example1", "example2"),
               dir = "~/tsdb/daily",
               columns = "A",
               return.class = "data.frame",
               drop.weekends = FALSE)
```

```
    timestamp ~/tsdb/daily/example1::A ~/tsdb/daily/example2::A
1  2016-01-01                        1                       11
2  2016-01-02                        2                       12
3  2016-01-03                        3                       13
4  2016-01-04                        4                       14
5  2016-01-05                        5                       15
6  2016-01-06                       NA                       16
7  2016-01-07                       NA                       17
8  2016-01-08                       NA                       18
9  2016-01-09                       NA                       19
10 2016-01-10                       NA                       20
```

The column names of the returned object consist of the filepaths and the column, which may be more information than we actually want. The argument `column.name` specifies the format; its default is `%dir%/%file%::%column%`.

```
read_ts_tables(c("example1", "example2"),
               dir = "~/tsdb/daily",
               columns = "A",
               return.class = "data.frame",
               drop.weekends = FALSE,
               column.name = "%file%/%column%")
```

```
    timestamp example1/A example2/A
1  2016-01-01          1         11
2  2016-01-02          2         12
3  2016-01-03          3         13
4  2016-01-04          4         14
5  2016-01-05          5         15
6  2016-01-06         NA         16
7  2016-01-07         NA         17
8  2016-01-08         NA         18
9  2016-01-09         NA         19
10 2016-01-10         NA         20
```

Missing values are by default set to `NA`. That happens even for missing columns, with a warning, though.

```
read_ts_tables(c("example1", "example2"),
               dir = "~/tsdb/daily",
               columns = c("A", "B"),
               return.class = "data.frame",
               drop.weekends = FALSE,
               column.name = "%file%/%column%")
```

```
    timestamp example1/A example1/B example2/A example2/B
1  2016-01-01          1         NA         11          1
2  2016-01-02          2         NA         12          2
3  2016-01-03          3         NA         13          3
4  2016-01-04          4         NA         14          4
5  2016-01-05          5         NA         15          5
6  2016-01-06         NA         NA         16          6
7  2016-01-07         NA         NA         17          7
8  2016-01-08         NA         NA         18          8
9  2016-01-09         NA         NA         19          9
10 2016-01-10         NA         NA         20         10
Warning message:
In read_ts_tables(c("example1", "example2"), dir = "~/tsdb/daily",  :
  columns missing
```

3 How tsdb works

3.1 The file format

tsdb can store and load time-series data. The format it uses is plain csv; a sample file my look as follows:

```
"timestamp","close"
17131,11
```

```
17132,12
17133,13
17134,14
17135,15
```

Thus, the file has a header line that gives the names of the columns, with the first column always being named `timestamp`.

The advantage of this plain format is that the data are in no way dependent on `tsdb`. The files can be used and manipulated by other software as well. For instance, if the above example is is stored in an appropriately named file `example`, we would reverse the order of the timestamps in a shell such as `bash`.

```
head -1 example; tail -n +2 example | sort -r
```

```
"timestamp","close"
17135,15
17134,14
17133,13
17132,12
17131,11
```

3.2 Timestamps