

# Package: aweek (via r-universe)

November 1, 2024

**Title** Convert Dates to Arbitrary Week Definitions

**Version** 1.0.2

**Description** Which day a week starts depends heavily on the either the local or professional context. This package is designed to be a lightweight solution to easily switching between week-based date definitions.

**Depends** R (>= 3.0)

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Suggests** testthat, stats, roxygen2, knitr, rmarkdown, covr, spelling

**RoxyenNote** 7.1.1

**Roxyen** list(markdown = TRUE)

**URL** <https://www.repidemicsconsortium.org/awee/>

**BugReports** <https://github.com/reconhub/awee/issues/>

**VignetteBuilder** knitr

**Language** en-US

**Repository** <https://reconhub.r-universe.dev>

**RemoteUrl** <https://github.com/reconhub/awee>

**RemoteRef** HEAD

**RemoteSha** 12785ec3623b8f50bdd24f81934e4a069734fb5b

## Contents

|                              |   |
|------------------------------|---|
| awee-package . . . . .       | 2 |
| as.awee . . . . .            | 4 |
| as.data.frame.awee . . . . . | 6 |
| as.Date.awee . . . . .       | 7 |
| change_week_start . . . . .  | 8 |

|                          |           |
|--------------------------|-----------|
| date2week . . . . .      | 9         |
| factor_aweeek . . . . .  | 12        |
| get_aweeek . . . . .     | 12        |
| print.aweeek . . . . .   | 15        |
| set_week_start . . . . . | 17        |
| <b>Index</b>             | <b>19</b> |

---

|                |  |
|----------------|--|
| aweeek-package | <i>Convert dates to weeks and back again</i> |
|----------------|--|

---

**Description**

The aweeek package is a lightweight solution for converting dates to weeks that can start on any weekday. It implements the [aweeek class](#), which can easily be converted to date and weeks that start on different days.

**Before you begin**

When you work with aweeek, you will want to make sure that you set the default week\_start variable to indicate which day of the week your weeks should begin. This can be done with [set\\_week\\_start\(\)](#). It will ensure that all of your weeks will begin on the same day.

- [get\\_week\\_start\(\)](#) returns the global week\_start option
- [set\\_week\\_start\(\)](#) sets the global week\_start option

**Conversions**

**Dates to weeks:**

This conversion is the simplest because dates are unambiguous.

- [date2week\(\)](#) converts dates, datetimes, and characters that look like dates to weeks
- [as.aweeek\(\)](#) is a wrapper around [date2week\(\)](#) that converts dates and datetimes

**Week numbers to weeks or dates:**

If you have separate columns for week numbers and years, then this is the option for you. This allows you to specify a different start for each week element using the start argument.

- [get\\_aweeek\(\)](#) converts week numbers (with years and days) to [aweeek objects](#).
- [get\\_date\(\)](#) converts week numbers (with years and days) to [Dates](#).

**ISO week strings (YYYY-Www-d or YYYY-Www) to weeks or dates:**

- [as.aweeek\(\)](#) converts ISO-week formatted strings to [aweeek objects](#).
- [week2date\(\)](#) converts ISO-week formatted strings to [Date](#).

**aweeek objects to dates or datetimes:**

This conversion is simple for [aweeek](#) objects since their week\_start is unambiguous

- [as.Date\(\)](#) converts to [Date](#).

- `as.POSIXlt()` converts to `POSIXlt`.

#### aweeek objects to characters:

You can strip the `week_start` attribute of the `aweeek` object by converting to a character with `as.character()`

#### Manipulating aweeek objects

- `trunc()` removes the weekday element of the ISO week string.
- `factor_aweeek()` does the same thing as `trunc()`, but will create a factor with levels spanning all the weeks from the first week to the last week. Useful for creating tables with zero counts for unobserved weeks.
- `change_week_start()` will change the `week_start` attribute and adjust the weeks accordingly so that the dates will always be consistent.

When you combine `aweeek` objects, they must have the same `week_start` attribute. Characters can be added to `aweeek` objects as long as they are in ISO week format and you can safely assume that they start on the same weekday. Dates are trivial to add to `aweeek` objects. See the [aweeek](#) documentation for details.

#### Author(s)

**Maintainer:** Zhian N. Kamvar <[zkamvar@gmail.com](mailto:zkamvar@gmail.com)>

#### See Also

Useful links:

- <https://www.repidemicsconsortium.org/aweeek/>
- Report bugs at <https://github.com/reconhub/aweeek/issues/>

#### Examples

```
# At the beginning of your analysis, set the week start to the weeks you want
# to use for reporting
ow <- set_week_start("Sunday")

# convert dates to weeks
d <- as.Date(c("2014-02-11", "2014-03-04"))
w <- as.aweeek(d)
w

# get the week numbers
date2week(d, numeric = TRUE)

# convert back to date
as.Date(w)

# convert to factor
factor_aweeek(w)
```

```
# append a week
w[3] <- as.Date("2014-10-31")
w

# change week start variable (if needed)
change_week_start(w, "Monday")

# note that the date remains the same
as.Date(change_week_start(w, "Monday"))

# Don't forget to reset the week_start at the end
set_week_start(ow)
```

---

as.aweek

---

*Convert characters or dates to aweek objects*


---

## Description

Convert characters or dates to aweek objects

## Usage

```
as.aweek(x, week_start = get_week_start(), ...)
```

## Default S3 method:

```
as.aweek(x, week_start = NULL, ...)
```

## S3 method for class 'NULL'

```
as.aweek(x, week_start = NULL, ...)
```

## S3 method for class 'character'

```
as.aweek(x, week_start = get_week_start(), start = week_start, ...)
```

## S3 method for class 'factor'

```
as.aweek(x, week_start = get_week_start(), ...)
```

## S3 method for class 'Date'

```
as.aweek(x, week_start = get_week_start(), ...)
```

## S3 method for class 'POSIXt'

```
as.aweek(x, week_start = get_week_start(), ...)
```

## S3 method for class 'aweeek'

```
as.aweek(x, week_start = NULL, ...)
```

## Arguments

x                    a [Date](#), [POSIXct](#), [POSIXlt](#), or a correctly formatted (YYYY-Www-d) character string that represents the year, week, and weekday.

|                         |  |
|-------------------------|--|
| <code>week_start</code> | a number indicating the start of the week based on the ISO 8601 standard from 1 to 7 where 1 = Monday OR an abbreviation of the weekday in an English or current locale. <i>Note: using a non-English locale may render your code non-portable.</i> Defaults to the value of <code>get_week_start()</code> |
| <code>...</code>        | arguments passed on to <code>date2week()</code> and <code>as.POSIXlt()</code>  |
| <code>start</code>      | an integer (or character) vector of days that the weeks start on for each corresponding week. Defaults to the value of <code>get_week_start()</code> . Note that these will not determine the final week.  |

## Details

The `as.aweek()` will coerce character, dates, and datetime objects to `aweek` objects. Dates are trivial to convert to weeks because there is only one correct way to convert them with any given `week_start`.

There is a bit of nuance to be aware of when converting characters to `aweek` objects:

- The characters must be correctly formatted as `YYYY-Www-d`, where `YYYY` is the year relative to the week, `Www` is the week number (`ww`) prepended by a `W`, and `d` (optional) is the day of the week from 1 to 7 where 1 represents the `week_start`. This means that characters formatted as dates will be rejected.
- By default, the `week_start` and `start` parameters are identical. If your data contains heterogeneous weeks (e.g. some dates will have the week start on Monday and some will have the week start on Sunday), then you should use the `start` parameter to reflect this. Internally, the weeks will first be converted to dates with their respective starts and then converted back to weeks, unified under the `week_start` parameter.

## Value

an `aweek` object

## Note

factors are first converted to characters before they are converted to `aweek` objects.

## See Also

`"aweek-class"` for details on the `aweek` object, `get_aweek()` for converting numeric weeks to weeks or dates, `date2week()` for converting dates to weeks, `week2date()` for converting weeks to dates.

## Examples

```
# aweek objects can only be created from valid weeks:

as.aweek("2018-W10-5", week_start = 7) # works!
try(as.aweek("2018-10-5", week_start = 7)) # doesn't work :(

# you can also convert dates or datetimes
as.aweek(Sys.Date())
as.aweek(Sys.time())
```

```
# all functions get passed to date2week, so you can use any of its arguments:
as.aweek("2018-W10-5", week_start = 7, floor_day = TRUE, factor = TRUE)
as.aweek(as.Date("2018-03-09"), floor_day = TRUE, factor = TRUE)

# If you have a character vector where different elements begin on different
# days of the week, you can use the "start" argument to ensure they are
# correctly converted.
as.aweek(c(mon = "2018-W10-1", tue = "2018-W10-1"),
         week_start = "Monday",
         start = c("Monday", "Tuesday"))

# you can convert aweek objects to aweek objects:
x <- get_aveek()
as.aweek(x)
as.aweek(x, week_start = 7)
```

---

as.data.frame.aweek     *Convert aweek objects to a data frame*

---

## Description

Convert aweek objects to a data frame

## Usage

```
## S3 method for class 'aveek'
as.data.frame(x, ...)
```

## Arguments

|     |                 |
|-----|-----------------|
| x   | an aweek object |
| ... | unused          |

## Value

a data frame with an aweek column

## See Also

[date2week\(\)](#) [print.aweek\(\)](#)

## Examples

```
d <- as.Date("2019-03-25") + 0:6
w <- date2week(d, "Sunday")
dw <- data.frame(date = d, week = w)
dw
dw$week
```

---

`as.Date.aweek`*Convert aweek objects to characters or dates*

---

## Description

Convert aweek objects to characters or dates

## Usage

```
## S3 method for class 'aweed'  
as.Date(x, floor_day = FALSE, ...)  
  
## S3 method for class 'aweed'  
as.POSIXlt(x, tz = "", floor_day = FALSE, ...)  
  
## S3 method for class 'aweed'  
as.character(x, ...)
```

## Arguments

|                        |   |
|------------------------|---|
| <code>x</code>         | an object of class <a href="#">aweed</a> .                |
| <code>floor_day</code> | when TRUE, the days will be set to the start of the week. |
| <code>...</code>       | parameters passed to <code>as.POSIXlt()</code> .          |
| <code>tz</code>        | passed on to <a href="#">as.POSIXlt()</a>                 |

## See Also

[date2week\(\)](#) [print.aweek\(\)](#)

## Examples

```
w <- date2week(Sys.Date(), week_start = "Sunday")  
w  
# convert to POSIX  
as.POSIXlt(w)  
as.POSIXlt(w, floor_day = TRUE)  
as.POSIXlt(w, floor_day = TRUE, tz = "KST")  
  
# convert to date  
as.Date(w)  
as.Date(w, floor_day = TRUE)  
  
# convert to character (strip attributes)  
as.character(w)
```

---

|                   |   |
|-------------------|---|
| change_week_start | <i>Change the week start of an aweek object</i> |
|-------------------|---|

---

## Description

This will change the week\_start attribute of an aweek object and adjust the observations accordingly.

## Usage

```
change_week_start(x, week_start = NULL, ...)
```

## Arguments

|            |  |
|------------|--|
| x          | a <a href="#">Date</a> , <a href="#">POSIXt</a> , <a href="#">character</a> , or any data that can be easily converted to a date with <a href="#">as.POSIXlt()</a> .   |
| week_start | a number indicating the start of the week based on the ISO 8601 standard from 1 to 7 where 1 = Monday OR an abbreviation of the weekday in an English or current locale. <i>Note: using a non-English locale may render your code non-portable.</i> Unlike <a href="#">date2week()</a> , this defaults to NULL, which will throw an error unless you supply a value. |
| ...        | arguments passed to <a href="#">as.POSIXlt()</a> , unused in all other cases.  |

## See Also

[get\\_week\\_start\(\)](#) for accessing the global and local week\_start attribute, [as.aweek\(\)](#), which wraps this function.

## Examples

```
# New Year's 2019 is the third day of the week starting on a Sunday
s <- date2week(as.Date("2019-01-01"), week_start = "Sunday")
s

# It's the second day of the week starting on a Monday
m <- change_week_start(s, "Monday")
m

# When you compare the underlying dates, they are exactly the same
identical(as.Date(s), as.Date(m))

# Since this will pass arguments to `date2week()`, you can modify other
# aspects of the aweek object this way, but this is not advised.

change_week_start(s, "Monday", floor_day = TRUE)
```



---

|           |   |
|-----------|---|
| date2week | <i>Convert date to a an arbitrary week definition</i> |
|-----------|---|

---

## Description

Convert date to a an arbitrary week definition

## Usage

```
date2week(
  x,
  week_start = get_week_start(),
  floor_day = factor,
  numeric = FALSE,
  factor = FALSE,
  ...
)

week2date(x, week_start = get_week_start(), floor_day = FALSE)
```

## Arguments

|            |   |
|------------|---|
| x          | a <a href="#">Date</a> , <a href="#">POSIXt</a> , <a href="#">character</a> , or any data that can be easily converted to a date with <a href="#">as.POSIXlt()</a> .  |
| week_start | a number indicating the start of the week based on the ISO 8601 standard from 1 to 7 where 1 = Monday OR an abbreviation of the weekday in an English or current locale. <i>Note: using a non-English locale may render your code non-portable.</i> Defaults to the value of <a href="#">get_week_start()</a>   |
| floor_day  | when TRUE, the days will be set to the start of the week.   |
| numeric    | if TRUE, only the numeric week be returned. If FALSE (default), the date in the format "YYYY-Www-d" will be returned.   |
| factor     | if TRUE, a factor will be returned with levels spanning the range of dates. This should only be used with floor_day = TRUE to produce the sequence of weeks between the first and last date as the factor levels. Currently, floor_date = FALSE will still work, but will produce a message indicating that it is deprecated. <i>Take caution when using this with a large date range as the resulting factor can contain all days between dates.</i> |
| ...        | arguments passed to <a href="#">as.POSIXlt()</a> , unused in all other cases.   |

## Details

Weeks differ in their start dates depending on context. The ISO 8601 standard specifies that Monday starts the week ([https://en.wikipedia.org/wiki/ISO\\_week\\_date](https://en.wikipedia.org/wiki/ISO_week_date)) while the US CDC uses Sunday as the start of the week ([https://www.cdc.gov/nndss/document/MMWR\\_Week\\_overview.pdf](https://www.cdc.gov/nndss/document/MMWR_Week_overview.pdf)). For example, MSF has varying start dates depending on country in order to better coordinate response.

While there are packages that provide conversion for ISOweeks and epiweeks, these do not provide seamless conversion from dates to epiweeks with non-standard start dates. This package provides a lightweight utility to be able to convert each day.

### Value

- `date2week()` an [aweek](#) object which represents dates in YYYY-Www-d format where YYYY is the year (associated with the week, not necessarily the day), Www is the week number prepended by a "W" that ranges from 01-53 and d is the day of the week from 1 to 7 where 1 represents the first day of the week (as defined by the `week_start` attribute).
- `week2date()` a [Date](#) object.

### Note

`date2week()` will initially convert the input with `as.POSIXlt()` and use that to calculate the week. If the user supplies character input, it is expected that the input will be of the format yyyy-mm-dd *unless* the user explicitly passes the "format" parameter to `as.POSIXlt()`. If the input is not in yyyy-mm-dd and the format parameter is not passed, `date2week()` will result in an error.

### Author(s)

Zhian N. Kamvar

### See Also

[set\\_week\\_start\(\)](#), [as.Date.aweek\(\)](#), [print.aweek\(\)](#), [as.aweek\(\)](#), [get\\_aweek\(\)](#)

### Examples

```
## Dates to weeks -----

# The same set of days will occur in different weeks depending on the start
# date. Here we can define a week before and after today

print(dat <- as.Date("2018-12-31") + -6:7)

# By default, the weeks are defined as ISO weeks, which start on Monday
print(iso_dat <- date2week(dat))

# This can be changed by setting the global default with set_week_start()

set_week_start("Sunday")

date2week(dat)

# If you want lubridate-style numeric-only weeks, you need look no further
# than the "numeric" argument
date2week(dat, numeric = TRUE)

# To aggregate weeks, you can use `floor_day = TRUE`
date2week(dat, floor_day = TRUE)
```

```

# If you want aggregations into factors that include missing weeks, use
# `floor_day = TRUE, factor = TRUE`:
date2week(dat[c(1, 14)], floor_day = TRUE, factor = TRUE)

## Weeks to dates -----

# The aweek class can be converted back to a date with `as.Date()`
as.Date(iso_dat)

# If you don't have an aweek class, you can use week2date(). Note that the
# week_start variable is set by the "awek.week_start" option, which we will
# set to Monday:

set_week_start("Monday")
week2date("2019-W01-1") # 2018-12-31

# This can be overridden by the week_start argument;
week2date("2019-W01-1", week_start = "Sunday") # 2018-12-30

# If you want to convert to the first day of the week, you can use the
# `floor_day` argument
as.Date(iso_dat, floor_day = TRUE)

## The same two week timespan starting on different days -----
# ISO week definition: Monday -- 1
date2week(dat, 1)
date2week(dat, "Monday")

# Tuesday -- 2
date2week(dat, 2)
date2week(dat, "Tuesday")

# Wednesday -- 3
date2week(dat, 3)
date2week(dat, "W") # you can use valid abbreviations

# Thursday -- 4
date2week(dat, 4)
date2week(dat, "Thursday")

# Friday -- 5
date2week(dat, 5)
date2week(dat, "Friday")

# Saturday -- 6
date2week(dat, 6)
date2week(dat, "Saturday")

# Epiweek definition: Sunday -- 7
date2week(dat, 7)
date2week(dat, "Sunday")

```

---

factor\_awareek

*Coerce an awareek object to factor to include missing weeks*


---

### Description

Coerce an awareek object to factor to include missing weeks

### Usage

```
factor_awareek(x)
```

### Arguments

x                      an awareek object

### Value

an awareek object that inherits from `factor()` with levels that span the range of the weeks in the object.

### Note

when factored awareek objects are combined with other awareek objects, they are converted back to characters.

### Examples

```
w <- get_awareek(week = (1:2) * 5, year = 2019, day = c(7, 1))
w
wf <- factor_awareek(w)
wf

# factors are destroyed if combined with awareek objects
c(w, wf)
```

---

get\_awareek

*Convert week numbers to dates or awareek objects*


---

### Description

These are vectorized functions that take integer vectors and return Date or an awareek objects, making it easier to convert bare weeks to dates.

**Usage**

```

get_aware(
  week = 1L,
  year = format(Sys.Date(), "%Y"),
  day = 1L,
  start = week_start,
  week_start = get_week_start(),
  ...
)

get_date(
  week = 1L,
  year = format(Sys.Date(), "%Y"),
  day = 1L,
  start = get_week_start()
)

```

**Arguments**

|            |   |
|------------|---|
| week       | an integer vector, defaults to 1, representing the first week of the year.  |
| year       | an integer vector, defaults to the current year   |
| day        | an integer vector, defaults to 1, representing the first day of the first week of the year.   |
| start      | an integer (or character) vector of days that the weeks start on for each corresponding week. Defaults to the value of <a href="#">get_week_start()</a> . Note that these will not determine the final week.  |
| week_start | a number indicating the start of the week based on the ISO 8601 standard from 1 to 7 where 1 = Monday OR an abbreviation of the weekday in an English or current locale. <i>Note: using a non-English locale may render your code non-portable.</i> Defaults to the value of <a href="#">get_week_start()</a> |
| ...        | parameters passed on to <a href="#">date2week()</a>   |

**Value**

- [get\\_aware\(\)](#): an aware object
- [get\\_date\(\)](#): a Date object

**Note**

Any missing weeks, years, or start elements will result in a missing element in the resulting vector. Any missing days will revert to the first day of the week.

**See Also**

[as.aware\(\)](#) [date2week\(\)](#) [week2date\(\)](#)

## Examples

```
# The default results in the first week of the year using the default
# default week_start (from get_week_start())

get_aware()
get_date() # this is equivalent to as.Date(get_week()), but faster

# Some years, like 2015, have 53 weeks

get_aware(53, 2015)

# If you specify 53 weeks for a year that doesn't have 53 weeks, aware will
# happily correct it for you

get_aware(53, 2014) # this will be 2015-W01-1

# you can use this to quickly make a week without worrying about formatting
# here, you can define an observation interval of 20 weeks

obs_start <- get_date(week = 10, year = 2018)
obs_end   <- get_date(week = 29, year = 2018, day = 7)
c(obs_start, obs_end)

# If you have a data frame of weeks, you can use it to convert easily

mat <- matrix(c(
  2019, 11, 1, 7, # 2019-03-10
  2019, 11, 2, 7,
  2019, 11, 3, 7,
  2019, 11, 4, 7,
  2019, 11, 5, 7,
  2019, 11, 6, 7,
  2019, 11, 7, 7
), ncol = 4, byrow = TRUE)

colnames(mat) <- c("year", "week", "day", "start")
m <- as.data.frame(mat)
m
sun <- with(m, get_date(week, year, day, start))
sun
as.aware(sun) # convert to aware starting on the global week_start
as.aware(sun, week_start = "Sunday") # convert to aware starting on Sunday

# You can also change starts
mon <- with(m, get_aware(week, year, day, "Monday", week_start = "Monday"))
mon
as.Date(mon)

# If you use multiple week starts, it will convert to date and then to
# the correct week, so it won't appear to match up with the original
# data frame.
```

```
sft <- with(m, get_aveek(week, year, day, 7:1, week_start = "Sunday"))
sft
as.Date(sft)
```

---

|             |                        |
|-------------|------------------------|
| print.aweek | <i>The aweek class</i> |
|-------------|------------------------|

---

## Description

The aweek class is a character or factor in the format YYYY-Www(-d) with a "week\_start" attribute containing an integer specifying which day of the ISO 8601 week each week should begin.

## Usage

```
## S3 method for class 'aweek'
print(x, ...)

## S3 method for class 'aweek'
x[i]

## S3 method for class 'aweek'
x[[i]]

## S3 replacement method for class 'aweek'
x[i] <- value

## S3 method for class 'aweek'
as.list(x, ...)

## S3 method for class 'aweek'
trunc(x, ...)

## S3 method for class 'aweek'
rep(x, ...)

## S3 method for class 'aweek'
c(..., recursive = FALSE, use.names = TRUE)
```

## Arguments

|                      |  |
|----------------------|--|
| x                    | an object of class aweek   |
| ...                  | a series of aweek objects, characters, or Dates, (unused in print.aweek()) |
| i                    | index for subsetting an aweek object.                                      |
| value                | a value to add or replace in an aweek object                               |
| recursive, use.names | parameters passed on to <a href="#">unlist()</a>                           |

## Details

Weeks differ in their start dates depending on context. The ISO 8601 standard specifies that Monday starts the week ([https://en.wikipedia.org/wiki/ISO\\_week\\_date](https://en.wikipedia.org/wiki/ISO_week_date)) while the US CDC uses Sunday as the start of the week ([https://www.cdc.gov/nndss/document/MMWR\\_Week\\_overview.pdf](https://www.cdc.gov/nndss/document/MMWR_Week_overview.pdf)). For example, MSF has varying start dates depending on country in order to better coordinate response.

While there are packages that provide conversion for ISOweeks and epiweeks, these do not provide seamless conversion from dates to epiweeks with non-standard start dates. This package provides a lightweight utility to be able to convert each day.

### Calculation of week numbers:

Week numbers are calculated in three steps:

1. Find the day of the week, relative to the week\_start (d). The day of the week (d) relative to the week start (s) is calculated using the ISO week day (i) via  $d = 1L + ((i + (7L - s)) \% 7L)$ .
2. Find the date that represents midweek (m). The date that represents midweek is found by subtracting the day of the week (d) from 4 and adding that number of days to the current date:  $m = \text{date} + (4 - d)$ .
3. Find the week number (w) by counting the number of days since 1 January to (m), and use integer division by 7:  $w = 1L + ((m - \text{yyyy-01-01}) \% 7)$

For the weeks around 1 January, the year is determined by the week number. If the month is January, but the week number is 52 or 53, then the year for the week (YYYY) is the calendar year (yyyy) minus 1. However, if the month is December, but the week number is 1, then the year for the week (YYYY) is the calendar year (yyyy) plus 1.

### Structure of the aweek object:

The aweek object is a character vector in either the precise ISO week format (YYYY-Www-d) or imprecise ISO week format (YYYY-Www) with a week\_start attribute indicating which ISO week day the week begins. The precise ISO week format can be broken down like this:

- **YYYY** is an ISO week-numbering year, which is the year relative to the week, not the day. For example, the date 2016-01-01 would be represented as 2015-W53-5 (ISO week), because while the date is in the year 2016, the week is still part of the final week of 2015.
- **Www** is the week number, prefixed by the character "W". This ranges from 01 to 52 or 53, depending on whether or not the year has 52 or 53 weeks.
- **d** is a digit representing the weekday where 1 represents the first day of the week and 7 represents the last day of the week. # The attribute week\_start represents the first day of the week as an ISO week day. This defaults to 1, which is Monday. If, for example, an aweek object represented weeks starting on Friday, then the week\_start attribute would be 5, which is Friday of the ISO week.

Imprecise formats (YYYY-Www) are equivalent to the first day of the week. For example, 2015-W53 and 2015-W53-1 will be identical when converted to date.

## Value

an object of class aweek



**Note**

when combining aweek objects together, you must ensure that they have the same week\_start attribute. You can use `change_week_start()` to adjust it.

**See Also**

`date2week()`, `get_aweek()`, `as.Date.aweek()`, `change_week_start()`

**Examples**

```
d <- as.Date("2018-12-20") + 1:40
w <- date2week(d, week_start = "Sunday")
print(w)

# subsetting acts as normal
w[1:10]

# Combining multiple aweek objects will only work if they have the same
# week_start day
c(w[1], w[3], w[5], as.aweek(as.Date("2018-12-01"), week_start = "Sunday"))

# differing week_start days will throw an error
mon <- date2week(as.Date("2018-12-01"), week_start = "Monday")
mon
try(c(w, mon))

# combining Dates will be coerced to aweek objects under the same rules
c(w, Sys.Date())

# truncated aweek objects will be un-truncated
w2 <- date2week(d[1:5], week_start = "Sunday", floor_day = TRUE)
w2
c(w[1:5], w2)
```

---

set\_week\_start

---

*Get and set the global week\_start variable*


---

**Description**

This is a convenience wrapper around `options()` and `getOption()`, which allows users to input both numeric and character week start values

**Usage**

```
set_week_start(x = 1L)
```

```
get_week_start(w = NULL)
```

**Arguments**

- |   |   |
|---|---|
| x | a character or integer specifying the day of the week for conversion between dates and weeks.                                     |
| w | if NULL, the global option "aweeek.week_start" is returned. If w is an aweek object, then the "week_start" attribute is returned. |

**Value**

for set\_week\_start, the old value of week\_start is returned, invisibly. For get\_week\_start, the current value of week\_start is returned.

**See Also**

[change\\_week\\_start\(\)](#) for changing the week\_start attribute of an aweek object, [date2week\(\)](#), [week2date\(\)](#)

**Examples**

```
# get the current definition of the week start
get_week_start() # defaults to Monday (1)
getOption("aweeek.week_start", 1L) # identical to above

# set the week start
mon <- set_week_start("Sunday") # set week start to Sunday (7)
get_week_start()
print(set_week_start(mon)) # reset the default
get_week_start()

# Get the week_start of a specific aweek object.
w <- date2week("2019-05-04", week_start = "Sunday")
get_week_start(w)
```

# Index

`[.aweeek (print.aweeek)`, 15  
`[<-.aweeek (print.aweeek)`, 15  
`[ [.aweeek (print.aweeek)`, 15  
  
`as.aweeek`, 4  
`as.aweeek()`, 2, 8, 10, 13  
`as.character()`, 3  
`as.character.aweeek (as.Date.aweeek)`, 7  
`as.data.frame.aweeek`, 6  
`as.Date()`, 2  
`as.Date.aweeek`, 7  
`as.Date.aweeek()`, 10, 17  
`as.list.aweeek (print.aweeek)`, 15  
`as.POSIXlt()`, 3, 5, 7–10  
`as.POSIXlt.aweeek (as.Date.aweeek)`, 7  
`aweeek`, 2, 3, 5, 7, 10  
`aweeek (aweeek-package)`, 2  
`aweeek class`, 2  
`aweeek objects`, 2  
`aweeek-class (print.aweeek)`, 15  
`aweeek-package`, 2  
  
`c.aweeek (print.aweeek)`, 15  
`change_week_start`, 8  
`change_week_start()`, 3, 17, 18  
`character`, 8, 9  
  
`Date`, 2, 4, 8–10  
`date2week`, 9  
`date2week()`, 2, 5–8, 13, 17, 18  
`Dates`, 2  
  
`factor()`, 12  
`factor_aweeek`, 12  
`factor_aweeek()`, 3  
  
`get_aweeek`, 12  
`get_aweeek()`, 2, 5, 10, 17  
`get_date (get_aweeek)`, 12  
`get_date()`, 2  
`get_week_start (set_week_start)`, 17  
  
`get_week_start()`, 2, 5, 8, 9, 13  
`getOption()`, 17  
  
`options()`, 17  
  
`POSIXct`, 4  
`POSIXlt`, 3, 4  
`POSIXt`, 8, 9  
`print.aweeek`, 15  
`print.aweeek()`, 6, 7, 10  
  
`rep.aweeek (print.aweeek)`, 15  
  
`set_week_start`, 17  
`set_week_start()`, 2, 10  
  
`trunc()`, 3  
`trunc.aweeek (print.aweeek)`, 15  
  
`unlist()`, 15  
  
`week2date (date2week)`, 9  
`week2date()`, 2, 5, 13, 18