Package 'date4ts'

April 29, 2026

Title Wrangle and Modify Ts Object with Classic Frequencies and Exact Dates

Version 0.1.2

Description The ts objects in R are managed using a very specific date format (in the form c(2022, 9) for September 2022 or c(2021, 2) for the second quarter of 2021, depending on the frequency, for example). We focus solely on monthly and quarterly series to manage the dates of ts objects. The general idea is to offer a set of functions to manage this date format without it being too restrictive or too imprecise depending on the rounding. This is a compromise between simplicity, precision and use of the basic 'stats' functions for creating and managing time series (ts(), window()).
Les objets ts en R sont gérés par un format de date très particulier (sous la forme c(2022, 9) pour septembre 2022 ou c(2021, 2) pour le deuxième trimestre 2021 selon la fréquence par exemple). On se concentre uniquement sur les séries mensuelles et trimestrielles pour gérer les dates des objets ts. Lidée générale est de proposer un ensemble de fonctions pour gérer ce format de date sans que ce soit trop contraignant ou trop imprécis selon les arrondis. Cest un compromis entre simplicité, précision et utilisation des fonctions du package 'stats' de création et de gestion des séries temporelles (ts(), window()).

License GPL (>= 3)

URL https://github.com/TractorTom/date4ts, https://tractortom.github.io/date4ts/

BugReports https://github.com/TractorTom/date4ts/issues

Encoding UTF-8

Language fr-FR

RoxygenNote 7.3.3

Imports stats, checkmate

Depends R (>= 4.1)

LazyData true

Suggests testthat (>= 3.0.0), renv, fuzzr, pkgdown, devtools, usethis, covr, withr, altdoc

Config/testthat/edition 3

NeedsCompilation no

Author Tanguy Barthelemy [aut, cre]

Maintainer Tanguy Barthelemy <tangbarth@hotmail.fr>

Repository CRAN

Date/Publication 2026-04-29 15:20:02 UTC

Contents

as_yyyytt . . . 3
check_date_ts . . . 4
check_expression . . . 5
check_frequency . . . 6
check_scalar_date . . . 8
check_scalar_integer . . . 9
check_scalar_natural . . . 10
check_timeunits . . . 12
check_ts . . . 13
combine2ts . . . 14
date2date_ts . . . 15
date_ts2date . . . 16
date_ts2timeunits . . . 17
diff_periode . . . 18
ev_pib . . . 19
extend_ts . . . 19
first_date . . . 21
get_value_ts . . . 22
is_before . . . 23
last_date . . . 24
libelles . . . 25
na_trim . . . 26
next_date_ts . . . 26
normalize_date_ts . . . 27
previous_date_ts . . . 28
set_value_ts . . . 29
substr_year . . . 30
trim2mens . . . 31
ts2df . . . 32

Index **33**

as_yyyytt *Conversion au format date_ts*

Description

Les fonctions as_yyyytt et as_yyyymm convertissent une date du format TimeUnits au format date_ts.

Usage

```
as_yyyytt(timeunits)

as_yyyymm(timeunits)
```

Arguments

timeunits	une date en année (Par exemple 2015.25 pour le 2ème trimestre 2015 ou 2021.83333333333 pour novembre 2021)

Details

La fonction as_yyyytt retourne la date par trimestre et la fonction as_yyyymm retourne la date par mois.

Value

En sortie, ces fonctions retournent la date au format date_ts (c'est-à-dire un vecteur d'entiers de la forme AAAA, c(AAAA, MM) ou c(AAAA, TT))

Examples

```
as_yyyytt(2019.75) # 4ème trimestre 2019
as_yyyytt(2020) # 1er trimestre 2020
as_yyyytt(2022 + 1 / 4) # 2ème trimestre 2022

as_yyyymm(2019.75) # Octobre 2019
as_yyyymm(2020) # Janvier 2020
as_yyyymm(2020 + 1 / 12) # Février 2020
as_yyyymm(2020 + 12 / 12) # Janvier 2021
```

check_date_ts *Vérifie le format de date*

Description

La fonction assert_date_ts vérifie qu'un objet est de type AAAA, c(AAAA, MM) ou c(AAAA, TT)

Usage

```
check_date_ts(x, frequency_ts, .var.name = checkmate::vname(x), warn = TRUE)

assert_date_ts(
  x,
  frequency_ts,
  add = NULL,
  .var.name = checkmate::vname(x),
  warn = TRUE
)
```

Arguments

x	un vecteur numérique, de préférence integer au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
frequency_ts	un entier qui vaut 4L (ou 4.0) pour les séries trimestrielles et 12L (ou 12.0) pour les séries mensuelles.
.var.name	Nom de l'objet à vérifier pour afficher dans les messages
warn	un booleen
add	Collection pour stocker les messages d'erreurs (Default is NULL)

Details

Les fonctions du package date4ts sont faites pour fonctionner avec des times-series de fréquence mensuelle ou trimestrielle et basés sur le système des mois, trimestres et années classiques. On cherche donc à favoriser l'utilisation de vecteur c(AAAA, MM) pour désigner la date choisie. Lorsque l'objet x en entrée est au mauvais format, il est corrigé pendant la checks et l'objet en sortie est au bon format. Si l'argument warn est FALSE, alors la fonction ne retournera pas de warning lors de l'évaluation.

Ici, l'argument frequency_ts est nécessaire car une date sous la forme c(AAAA, PP), avec PP le nombre de période, ne désigne pas une date absolue. Par exemple, c(2020L 5L) désigne mai 2020 pour une fréquence mensuelle et le 1er trimestre 2021 pour une fréquence trimestrielle.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction assert_date_ts retourne l'objet x de manière invisible;
  - la fonction check_date_ts retourne le booléen TRUE.

- si le check échoue :
  - la fonction `assert_date_ts` retourne un message d'erreur;
  - la fonction `check_date_ts` retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet `x` de manière invisible ou une erreur.

Examples

```
# De bons formats de date
assert_date_ts(c(2020L, 8L), frequency_ts = 12L)
assert_date_ts(c(2020L, 2L), frequency_ts = 4L)
check_date_ts(2022L, frequency_ts = 12L)

# Format double --> génération d'un warning
assert_date_ts(c(2020., 4.0), frequency_ts = 4L)
assert_date_ts(2022., frequency_ts = 12L)
check_date_ts(2022., frequency_ts = 12L)

# Fréquence au format double --> génération d'un warning
assert_date_ts(c(2020L, 6L), frequency_ts = 4.0)
assert_date_ts(c(2020L, 42L), frequency_ts = 12.0)

# Dépassement la fréquence --> génération d'un warning
assert_date_ts(c(2020L, 6L), frequency_ts = 4L)
assert_date_ts(c(2020L, 42L), frequency_ts = 12L)
assert_date_ts(c(2020L, -4L), frequency_ts = 12L)

# Avec des erreurs
check_date_ts(1:10, frequency_ts = 12L)
```

---

check_expression *Vérifie la conformité d'une expression*

Description

Vérifie la conformité d'une expression

Usage

```
check_expression(expr)

assert_expression(expr)
```

Arguments

`expr`	une expression à évaluer

Details

La fonction évalue l'expression `expr`. Le check vérifie si la fonction génère une erreur ou un warning. Si elle ne génère aucun message particulier, on retourne alors l'objet `x` (le résultat de l'évaluation de l'expression `expr`), sans erreur.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_expression` retourne l'objet `x` de manière invisible;
  - la fonction `check_expression` retourne le booléen `TRUE`.
- si le check échoue :
  - la fonction `assert_expression` retourne un message d'erreur;
  - la fonction `check_expression` retourne la chaîne de caractère "Invalid expression".

Value

En sortie la fonction retourne l'objet `x` (le résultat de l'évaluation de l'expression `expr`) de manière invisible ou une erreur.

Examples

```
assert_expression(expr = {2 + 2})
assert_expression(expr = {is.integer(1L)})
try(assert_expression(expr = {log("a")}), silent = TRUE)

check_expression(expr = {2 + 2})
check_expression(expr = {is.integer(1L)})
check_expression(expr = {log("a")})
```

---

check_frequency *Vérifie la conformité d'une fréquence*

Description

Vérifie la conformité d'une fréquence

Usage

```
check_frequency(x, .var.name = checkmate::vname(x), warn = TRUE)

assert_frequency(x, add = NULL, .var.name = checkmate::vname(x), warn = TRUE)
```

Arguments

x	un entier qui vaut 4L (ou 4.0) pour les séries trimestrielles et 12L (ou 12.0) pour les séries mensuelles.
.var.name	Nom de l'objet à vérifier pour afficher dans les messages
warn	un booleen
add	Collection pour stocker les messages d'erreurs (Default is NULL)

Details

La fréquence d'une série temporelle est soit mensuelle (12L ou 12.0) soit trimestrielle (4L ou 4.0). Les autres fréquences ne sont pas acceptées. Cette fonction s'appuie essentiellement sur les fonctions checkmate::check_numeric, checkmate::check_int et checkmate::check_choice. Il y a néanmoins une petite subtilité : on vérifie si l'objet x est de type double ou integer. Dans le premier cas, on affichera un warning et on corrigera l'objet au format integer pour les traitements ultérieurs. En sortie, x est retourné de manière invisible. Si l'argument warn est FALSE, alors la fonction ne retournera pas de warning lors de l'évaluation.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction assert_frequency retourne l'objet x de manière invisible;
  - la fonction check_frequency retourne le booléen TRUE.
- si le check échoue :
  - la fonction assert_frequency retourne un message d'erreur;
  - la fonction check_frequency retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet x de manière invisible ou une erreur.

Examples

```
assert_frequency(4L)
assert_frequency(12L)

check_frequency(4L)
check_frequency(12L)

# Avec des erreurs,

check_frequency(Inf, warn = FALSE)
check_frequency(1:10)
check_frequency(1L)
```

`check_scalar_date` *Vérifie la conformité d'une date scalaire*

Description

Vérifie la conformité d'une date scalaire

Usage

```
check_scalar_date(x)

assert_scalar_date(x, add = NULL, .var.name = checkmate::vname(x))
```

Arguments

`x`	un objet de type `Date`.
`add`	Collection pour stocker les messages d'erreurs (Default is NULL)
`.var.name`	Nom de l'objet à vérifier pour afficher dans les messages

Details

On vérifie que l'objet `x` en entrée est bien au format `Date` et qu'il s'agit d'un scalaire (vecteur de taille 1). Cette fonction s'appuie essentiellement sur la fonction `checkmate::assert_date`.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_scalar_date` retourne l'objet `x` de manière invisible;
  - la fonction `check_scalar_date` retourne le booléen `TRUE`.
- si le check échoue :
  - la fonction `assert_scalar_date` retourne un message d'erreur;
  - la fonction `check_scalar_date` retourne la chaîne de caractère correspondante à l'erreur du check.

Value

En sortie la fonction retourne l'objet `x` de manière invisible ou une erreur.

Examples

```
assert_scalar_date(as.Date("2018-01-24"))
assert_scalar_date(as.Date("2000-02-29"))
assert_scalar_date(Sys.Date())

check_scalar_date(as.Date("2018-01-24"))
check_scalar_date(as.Date("2000-02-29"))
check_scalar_date(Sys.Date())
```

```
# Avec des erreurs

check_scalar_date(2L)
check_scalar_date(seq(from = as.Date("2000-01-01"), to = Sys.Date(), by =
"year"))
```

`check_scalar_integer` *Vérifie la conformité d'un entier scalaire*

Description

Vérifie la conformité d'un entier scalaire

Usage

```
check_scalar_integer(x, warn = TRUE)

assert_scalar_integer(
  x,
  add = NULL,
  .var.name = checkmate::vname(x),
  warn = TRUE
)
```

Arguments

`x`	un entier relatif (positif, négatif ou nul)
`warn`	un booleen
`add`	Collection pour stocker les messages d'erreurs (Default is NULL)
`.var.name`	Nom de l'objet à vérifier pour afficher dans les messages

Details

On vérifie que l'objet `x` en entrée est bien un entier. Cette fonction s'appuie essentiellement sur la fonction `checkmate::assert_int`. Il y a néanmoins une petite subtilité : on vérifie si l'objet `x` est de type double ou integer. Si l'objet est de type double (et non integer), la fonction retournera aussi un warning. Dans le premier cas, on affichera un warning et on corrigera l'objet au format integer pour les traitements ultérieurs. En sortie, `x` est retourné de manière invisible. Si l'argument `warn` vaut `FALSE`, alors la fonction ne retournera pas de warning lors de l'évaluation.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_scalar_integer` retourne l'objet `x` de manière invisible;
  - la fonction `check_scalar_integer` retourne le booléen `TRUE`.

- si le check échoue :
  - la fonction `assert_scalar_integer` retourne un message d'erreur;
  - la fonction `check_scalar_integer` retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet x de manière invisible ou une erreur.

See Also

`check_scalar_natural()`, `assert_scalar_natural()`

Examples

```
assert_scalar_integer(1L)
assert_scalar_integer(100L)
assert_scalar_integer(-4L)
assert_scalar_integer(0L)

check_scalar_integer(1L)
check_scalar_integer(100L)
check_scalar_integer(-4L)
check_scalar_integer(0L)

# Avec des erreurs,

check_scalar_integer(Inf)
check_scalar_integer(1:10)
check_scalar_integer(pi)
check_scalar_integer(2.)
```

---

check_scalar_natural *Vérifie la conformité d'un entier naturel*

Description

Le but de cett fonction est de tester si une variable x est un nombre naturel strictement positif.

Usage

```
check_scalar_natural(x, warn = TRUE)

assert_scalar_natural(
  x,
  add = NULL,
  .var.name = checkmate::vname(x),
  warn = TRUE
)
```

Arguments

x	un entier naturel strictement positif
warn	un booleen
add	Collection pour stocker les messages d'erreurs (Default is NULL)
.var.name	Nom de l'objet à vérifier pour afficher dans les messages

Details

Cette fonction s'appuie essentiellement sur la fonction `checkmate::assert_count`. Il y a néanmoins une petite subtilité : on vérifie si l'objet `x` est de type double ou integer. Dans le premier cas, on affichera un warning et on corrigera l'objet au format integer pour les traitements ultérieurs. En sortie, `x` est retourné de manière invisible. Si l'argument `warn` est `FALSE`, alors la fonction ne retournera pas de warning lors de l'évaluation.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_scalar_natural` retourne l'objet `x` de manière invisible;
  - la fonction `check_scalar_natural` retourne le booléen `TRUE`.
- si le check échoue :
  - la fonction `assert_scalar_natural` retourne un message d'erreur;
  - la fonction `check_scalar_natural` retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet `x` de manière invisible ou une erreur.

See Also

`check_scalar_integer()`, `assert_scalar_integer()`

Examples

```
# Avec des entier integer
assert_scalar_natural(1L)
assert_scalar_natural(100L)

# Avec des entiers double
assert_scalar_natural(2.)
assert_scalar_natural(457)
```

`check_timeunits` *Vérifie la conformité d'un objet TimeUnits*

Description

La fonction `assert_timeunits` vérifie qu'un objet est un TimeUnits.

Usage

```
check_timeunits(x, frequency_ts, .var.name = checkmate::vname(x))

assert_timeunits(x, frequency_ts, add = NULL, .var.name = checkmate::vname(x))
```

Arguments

`x`	un numérique qui représente le time units de
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.
`.var.name`	Nom de l'objet à vérifier pour afficher dans les messages
`add`	Collection pour stocker les messages d'erreurs (Default is NULL)

Details

Un objet de type TimeUnits est un numérique qui désigne l'année et la période en cours avec ses décimales. Ainsi pour une série temporelle mensuelle, `2020.5` représente la moitié de l'année donc juillet 2020 et s'écrit `c(2020L, 7L)` au format date_ts.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_timeunits` retourne l'objet `x` de manière invisible;
  - la fonction `check_timeunits` retourne le booléen `TRUE`.
- si le check échoue :
  - la fonction `assert_timeunits` retourne un message d'erreur;
  - la fonction `check_timeunits` retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet `x` de manière invisible ou une erreur.

Examples

```
assert_timeunits(2020.5, frequency_ts = 12L)
assert_timeunits(2020.5, frequency_ts = 4L)
assert_timeunits(2023., frequency_ts = 12L)

assert_timeunits(2000. + 5. / 12.0, frequency_ts = 12L)
assert_timeunits(2015. + 3. / 4.0, frequency_ts = 4L)

check_timeunits(2020.5, frequency_ts = 12L)
check_timeunits(2015. + 3. / 4.0, frequency_ts = 4L)

# Avec erreur

check_timeunits(list(1.), frequency_ts = 12L)
check_timeunits(2000., frequency_ts = 1L)
```

---

check_ts *Vérifie la conformité d'un objet ts*

Description

Les fonctions `assert_ts` et `check_ts` vérifient qu'un objet ts est bien conforme.

Usage

```
check_ts(x, .var.name = checkmate::vname(x), allow_mts = FALSE)

assert_ts(x, add = NULL, .var.name = checkmate::vname(x), allow_mts = FALSE)
```

Arguments

`x`	Un objet ts unidimensionnel
`.var.name`	Nom de l'objet à vérifier pour afficher dans les messages
`allow_mts`	Booleen. Est ce que les objects `mts` sont acceptés ?
`add`	Collection pour stocker les messages d'erreurs (Default is NULL)

Details

Les fonctions du package date4ts sont faites pour fonctionner avec des times-series de fréquence mensuelle ou trimestrielle et basées sur le système des mois, trimestres et années classiques. On travaille avec des données numériques (integer, double ou logical) mais les autres types atomic sont acceptés également. On cherche donc à favoriser l'utilisation de séries temporelles classiques utilisants des types atomiques. Lorsque l'objet `x` en entrée est au mauvais format, une erreur est généré.

Selon le préfixe de la fonction :

- si le check réussi :
  - la fonction `assert_ts` retourne l'objet `x` de manière invisible;
  - la fonction `check_ts` retourne le booléen `TRUE`.
- si le check échoue :
  - la fonction `assert_ts` retourne un message d'erreur;
  - la fonction `check_ts` retourne une chaîne de caractère signalant le problème.

Value

En sortie la fonction retourne l'objet `x` de manière invisible ou une erreur.

Examples

```
ts1 <- ts(1:100, start = 2010L, frequency = 12L)
ts2 <- ts(1:10, start = c(2020L, 4L), frequency = 4L)

assert_ts(ts1)
assert_ts(ts2)

check_ts(ts1)
check_ts(ts2)

# Exemples avec des erreurs

check_ts(1)
check_ts(ts(1:10, start = 2010L, frequency = 2L))
check_ts(1:10)
```

`combine2ts` *Combiner 2 ts*

Description

La fonction `combine2ts` combine (comme c()) 2 time series de même fréquence (mensuelle ou trimestrielle).

Usage

```
combine2ts(a, b)
```

Arguments

`a`	un objet ts unidimensionnel conforme aux règles de assert_ts
`b`	un objet ts unidimensionnel conforme aux règles de assert_ts

Details

Si a et b ont une période en commun, les valeurs de b écrasent celles de a sur la période concernée. Si il existe une période sur laquelle ni a ni b ne prennent de valeur (mais qu'il existe des valeurs à des dates ultérieures et antérieures) alors le ts en sortie prendra NA sur cette période.

Value

En sortie, la fonction retourne un ts qui contient les valeurs de a aux temps de a et les valeurs de b aux temps de b.

Examples

```
trim_1 <- stats::ts(rep(1, 4), start = 2021, frequency = 4L)

mens_1 <- stats::ts(rep(1, 4), start = 2020, frequency = 12L)
mens_2 <- stats::ts(rep(2, 4), start = 2022, frequency = 12L)

# La série de PIB est écrasé par trim_1 sur la période temporelle de trim_1
combine2ts(ev_pib, trim_1)

# La période entre les séries temporelles mens_1 et mens_2 est complétée par
# des NA
combine2ts(mens_1, mens_2)
```

---

date2date_ts *Conversion d'une date au format TS*

Description

La fonction `date2date_ts` prend en argument une date au format date (integer avec une class Date) et la convertit au format `date_ts` : `c(AAAA, MM)` ou `c(AAAA, TT)` avec le mois ou trimestre en cours.

Usage

```
date2date_ts(date, frequency_ts = 12L)
```

Arguments

`date`	un objet de type Date
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.

Value

En sortie, la fonction retourne la date au format `date_ts` (`c(AAAA, MM)` ou `c(AAAA, TT)`) avec le mois ou trimestre en cours selon l'argument `frequency_ts`.

Examples

```
date2date_ts(as.Date("2000-01-01"))
date2date_ts(as.Date("2000-01-01"), frequency_ts = 12L)

date2date_ts(as.Date("2021-10-01"), frequency_ts = 12L)
date2date_ts(as.Date("2021-10-01"), frequency_ts = 4L)
```

`date_ts2date` *Conversion d'une date du format TS au format date*

Description

Conversion d'une date du format TS au format date

Usage

```
date_ts2date(date_ts, frequency_ts)
```

Arguments

`date_ts`	un vecteur numérique, de préférence `integer`, au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.

Value

En sortie, la fonction retourne un objet de type Date (atomic) de longueur 1 qui correspond à l'objet `date_ts`.

Examples

```
date_ts2date(date_ts = c(2020L, 11L), frequency_ts = 12L)
date_ts2date(date_ts = c(1995L, 2L), frequency_ts = 4L)
```

`date_ts2timeunits` *Conversion d'une date du format date_ts au format TimeUnits*

Description

Conversion d'une date du format date_ts au format TimeUnits

Usage

```
date_ts2timeunits(date_ts, frequency_ts)
```

Arguments

`date_ts`	un vecteur numérique, de préférence `integer`, au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.

Details

`AAAA` signifie que l'année est au format numérique avec 4 chiffres (Exemple : l'année deux mille vingt-deux s'écrit 2022 et non 22) MM signifie que le mois est au format numérique (Exemple : le mois de mai s'écrit 5, le moi de décembre s'écrit 12) TT signifie que le trimestre est au format numérique (Exemple : le troisième trimestre s'écrit 3)

Value

En sortie, la fonction retourne la date au format `AAAA + TT/4` ou `AAAA + MM/12` (un numeric de longueur 1).

Examples

```
# Avril 2020
date_ts2timeunits(date_ts = c(2020L, 4L), frequency_ts = 12L)
# Novembre 2020
date_ts2timeunits(date_ts = c(2022L, 11L), frequency_ts = 12L)

# 4ème trimestre de 2022
date_ts2timeunits(date_ts = c(2022, 4L), frequency_ts = 4L)
# 2ème trimestre de 1995
date_ts2timeunits(date_ts = c(1995L, 2L), frequency_ts = 4L)
```

`diff_periode` *Intervalle entre 2 dates*

Description

Intervalle entre 2 dates

Usage

```
diff_periode(a, b, frequency_ts)
```

Arguments

`a`	un objet date_ts, c'est-à-dire un vecteur numérique, de préférence `integer` au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`b`	un objet date_ts, c'est-à-dire un vecteur numérique, de préférence `integer` au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.

Details

On travaille ici avec des dates au format date_ts, c'est-à-dire qui passe le test de la fonction `assert_date_ts`. Lorsqu'on parle d'intervalle et de nombre de période entre `a` et `b`, les bornes sont incluses. Ainsi `diff_periode(2020L, 2020L, 12L)` retourne bien 1L et non 2L ou 0L.

Value

En sortie, la fonction retourne un entier qui désigne le nombre de période (mois ou trimestres) qui sépare les 2 dates `a` et `b`.

Examples

```
# Une seule période
diff_periode(a = 2020L, b = 2020L, frequency_ts = 4L)

diff_periode(a = c(2000L, 1L), b = c(2020L, 4L), frequency_ts = 4L)

# Ordre chronologique respecté
diff_periode(a = c(2021L, 5L), b = c(2023L, 8L), frequency_ts = 12L)

# Date inversées
diff_periode(a = c(2023L, 8L), b = c(2021L, 5L), frequency_ts = 12L)
```

ev_pib *Évolution du PIB français jusqu'au T1 2022*

Description

Ce jeu de données contient une série `ts` de l'évolution trimestrielle du produit intérieur brut français. Toutes les infos complémentaires sur cette série se trouve sur la page de la publication sur le site de l'Insee.

Usage

```
ev_pib
```

Format

Un ts unidimensionnel :

start le ts commence au T1 1970 mais la série de PIB ne commence qu'au T2 1980.

end le ts finit au T3 2022 mais la série de PIB finit au T1 2022.

frequency_ts la fréquence est trimestrielle

Source

https://www.insee.fr/fr/statistiques/2830547

extend_ts *Ajoute de nouvelles valeurs à un ts*

Description

La fonction `extend_ts` ajoute de nouvelles valeurs à un ts.

Usage

```
extend_ts(
  series,
  replacement,
  date_ts_to = NULL,
  replace_na = TRUE,
  times = 1L,
  each = 1L
)
```

Arguments

series	un objet ts unidimensionnel conforme aux règles de assert_ts
replacement	un vecteur de même type que le ts series
date_ts_to	un vecteur numérique, de préférence integer, au format date_ts, c'est-à-dire AAAA, c(AAAA, MM) ou c(AAAA, TT).
replace_na	un booléen.
times	un entier qui précise le nombre de fois où replacement doit être répété, le vecteur entier.
each	un entier qui précise le nombre de fois où replacement doit être répété mais élément par élément.

Details

date_ts_to désigne la date jusqu'à laquelle le remplacement s'effectue. Par défault, cette valeur vaut NULL.

Si replace_na vaut TRUE alors le remplacement commence dès que l'objet ne contient que des NA. Dans le cas contraire, le ts est étendu, qu'il contienne des NA ou non à la fin. Si le vecteur replacement est de taille un sous-multiple de la différence de période entre la date de fin de series et date_ts_to, le vecteur replacement est répété jusqu'à la date date_ts_to. Sinon une erreur est générée.

Les arguments times et each en sont utilisé que si date_ts est manquant (non fourni par l'utilisateur). Si tel est le cas, ils se comporte comme si replacement devenait rep(replacement, times = times, each = each).

Value

En sortie, la fonction retourne une copie de l'objet series complété avec le vecteur replacement.

Examples

```
ts1 <- ts(
    data = c(rep(NA_integer_, 3L), 1L:10L, rep(NA_integer_, 3L)),
    start = 2020,
    frequency = 12
)
x <- rep(3L, 2L)

extend_ts(series = ts1, replacement = x)
extend_ts(series = ts1, replacement = x, replace_na = FALSE)
extend_ts(series = ts1, replacement = x,
          date_ts_to = c(2021L, 7L), replace_na = TRUE)
```

`first_date` *Première date non NA*

Description

Cette fonction calcule la première date pour laquelle l'objet `series` ne vaut pas NA.

Usage

```
first_date(series)
```

Arguments

`series`	un objet ts unidimensionnel conforme aux règles de assert_ts

Details

La date retournée en output est au format `date_ts`. Si l'objet `series` ne contient que des NAs, la fonction retourne une erreur.

Value

En sortie, la fonction retourne un objet au format `date_ts` (`AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`)

See Also

`last_date`

Examples

```
ts1 <- ts(c(NA, NA, NA, 1:10, NA), start = 2000, frequency = 12L)
ts2 <- ts(c(1:10, NA), start = 2020, frequency = 4L)

stats::start(ts1)
first_date(ts1)

stats::start(ts1)
first_date(ts2)
```

get_value_ts *Récupère des valeurs d'un ts*

Description

La fonction get_value_ts permet de récupérer des valeurs.

Usage

```
get_value_ts(series, date_from, date_to, n)
```

Arguments

series	un objet ts unidimensionnel conforme aux règles de assert_ts
date_from	un vecteur numérique, de préférence integer au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
date_to	un vecteur numérique, de préférence integer au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
n	un entier

Details

Il faut qu'exactement 2 arguments parmi date_to, date_to et n soient renseignés. L'argument n combiné avec date_to ou date_from permet de déterminer combien de période seront retourné à partir de ou jusqu'à la date renseignée.

Il faudrait parler d'extraction car contrairement à la fonction window, ici on retourne un vecteur devaleur et plus un objet ts.

Value

En sortie, la fonction retourne un vecteur (atomic) de même type que series avec les valeurs extraites.

Examples

```
ts1 <- ts(1:100, start = 2012L, frequency = 12L)
ts2 <- ts(letters, start = 2014L, frequency = 4L)
ts3 <- ts(exp(-(1:50)), start = 2015L, frequency = 12L)

get_value_ts(series = ts1, date_from = c(2015L, 7L), date_to = c(2018L, 6L))
get_value_ts(series = ts2, date_from = c(2018L, 4L), n = 4L)
get_value_ts(series = ts3, date_to = c(2018L, 4L), n = 14L)
```

`is_before` *Comparaison de 2 date_ts*

Description

Comparaison de 2 date_ts

Usage

```
is_before(a, b, frequency_ts, strict = FALSE)
```

Arguments

`a`	un objet date_ts, c'est-à-dire un vecteur numérique, de préférence `integer` au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`b`	un objet date_ts, c'est-à-dire un vecteur numérique, de préférence `integer` au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.
`strict`	un booleen (default `FALSE`)

Details

Les dates `a` et `b` sont au format date_ts. L'argument frequency_ts est nécessaire pour interpréter les dates. Ainsi, si je souhaite comparer la date `a = c(2023L, 4L)` et la date `b = c(2023L, -2L)`. Dans le cas d'une fréquence mensuelle, la date `a` est antérieure à la date `b`. Dans le cas d'une fréquence mensuelle, c'est l'inverse. Si `strict` vaut `TRUE`, la fonction compare strictement les dates `a` et `b` (<).

Value

En sortie, la fonction retourne un booleen (de longueur 1) qui indique si la date `a` est antérieure à la date `b`.

Examples

```
is_before(a = c(2020L, 3L), b = c(2022L, 4L), frequency_ts = 12L)
is_before(a = c(2022L, 3L), b = c(2010L, 1L), frequency_ts = 4L)

is_before(a = c(2022L, 4L), b = c(2022L, 4L), frequency_ts = 12L)
is_before(a = c(2022L, 4L), b = c(2022L, 4L),
    frequency_ts = 12L, strict = TRUE)

# Importance de la fréquence
is_before(a = c(2022L, -3L), b = c(2021L, 8L), frequency_ts = 12L)
is_before(a = c(2022L, -3L), b = c(2021L, 8L), frequency_ts = 4L)
```

`last_date` *Dernière date non NA*

Description

Cette fonction calcule la dernière date pour laquelle l'objet `series` ne vaut pas NA.

Usage

```
last_date(series)
```

Arguments

`series`	un objet ts unidimensionnel conforme aux règles de assert_ts

Details

La date retournée en output est au format `date_ts`. Si l'objet `series` ne contient que des NAs, la fonction retourne une erreur.

Value

En sortie, la fonction retourne un objet au format `date_ts` (`AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`)

See Also

`first_date`

Examples

```
ts1 <- ts(c(NA, NA, NA, 1:10, NA), start = 2000, frequency = 12L)
ts2 <- ts(c(1:10), start = 2020, frequency = 4L)

stats::end(ts1)
last_date(ts1)

stats::end(ts2)
last_date(ts2)
```

`libelles` *Libelés pour une période*

Description

La fonction `libelles` créé un vecteur de chaines de caractère contenant les libelés de toutes les dates sur une période

Usage

```
libelles(date_ts, frequency_ts, n = 1L, warn = TRUE)
```

Arguments

`date_ts`	un vecteur numérique, de préférence `integer`, au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`
`frequency_ts`	un entier qui vaut `4L` (ou `4.0`) pour les séries trimestrielles et `12L` (ou `12.0`) pour les séries mensuelles.
`n`	un entier
`warn`	un booleen

Details

Pour choisir la période, il faut spécifier une date de début `date_ts`, une fréquence `frequency_ts` pour le pas entre 2 dates (trimestrielle ou mensuelle) et un nombre de valeur `n` (nombre de période).

Si l'argument `warn` est `FALSE`, alors la fonction ne retournera pas de warning lors de l'évaluation.

Value

En sortie, la fonction retourne un vecteur de chaine de caractère de longueur `n` avec les libellés de la période (de la date `date_ts` à la date `date_ts + n périodes`.

Examples

```
libelles(date_ts = c(2019L, 10L), frequency_ts = 12L, n = 9L)
libelles(date_ts = c(2019L, 4L), frequency_ts = 4L, n = 3L)
```

na_trim *Supprime les NA aux bords*

Description

La fonction na_trim supprime les NA en début et en fin de période.

Usage

```
na_trim(series, sides = c("both", "left", "right"))
```

Arguments

series	un objet ts unidimensionnel conforme aux règles de assert_ts
sides	une chaine de caractere qui spécifie quelle NA doivent être retirés (au début et à la fin ("both"), juste au début ("left") ou juste à la fin ("right"))

Details

L'objet retourné commence et finis par des valeurs non manquantes.

Value

En sortie, la fonction retourne une copie de l'objet series corrigée des NA et début et fin de série.

Examples

```
ts1 <- ts(c(rep(NA, 3L), 1:10, rep(NA, 3L)), start = 2020, frequency = 12L)
ts2 <- ts(c(1:10, rep(NA, 3L)), start = c(2023, 2), frequency = 4L)
ts3 <- ts(c(rep(NA, 3L), 1:10), start = 2000, frequency = 12L)

na_trim(ts1)
na_trim(ts2)
na_trim(ts3)
```

next_date_ts *Obtenir la date suivante*

Description

Obtenir la date suivante

Usage

```
next_date_ts(date_ts, frequency_ts, lag = 1L)
```

Arguments

date_ts	un vecteur numérique, de préférence integer, au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
frequency_ts	un entier qui vaut 4L (ou 4.0) pour les séries trimestrielles et 12L (ou 12.0) pour les séries mensuelles.
lag	un entier

Details

Lorsqu'on parle de date suivante, on parle de date future. L'argument lag est entier et désigne le nombre de décalage que l'on affecte à notre date. Par exemple pour des lag positif (1L, 2L, 10L) on désigne le décalage de la période suivante, celle d'après et celle dans 10 périodes. Cependant, lorsque l'argument lag vaut zéro, la fonction retourne la date inchangée. Aussi lorsque l'argument lag est négatif, la fonction se comporte comme la fonction previous_date_ts et retourne les périodes passées et non futures.

Value

En sortie, la fonction retourne un vecteur d'entier qui représente la date à la période future au format date_ts.

See Also

previous_date_ts

Examples

```
next_date_ts(c(2020L, 4L), frequency_ts = 4L, lag = 2L)
next_date_ts(c(2021L, 1L), frequency_ts = 4L, lag = -2L)

next_date_ts(c(2020L, 4L), frequency_ts = 12L, lag = 2L)
next_date_ts(c(2022L, 6L), frequency_ts = 12L, lag = 12L)
```

---

normalize_date_ts *Ajuste un objet date_ts dans un format conforme.*

Description

Ajuste un objet date_ts dans un format conforme.

Usage

```
normalize_date_ts(date_ts, frequency_ts, test = TRUE)
```

Arguments

date_ts	un vecteur numérique, de préférence integer, au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
frequency_ts	un entier qui vaut 4L (ou 4.0) pour les séries trimestrielles et 12L (ou 12.0) pour les séries mensuelles.
test	un booléen (Default is TRUE)

Details

Ici le formattage correspond à une réécriture de la date sans en changer la valeur. Alors que l'objet c(2020L, 12L) désigne le mois de décembre 2020 et c(2021L, 1L) le mois de janvier 2021, on peut imaginer que la date_ts c(2021L, 0L) peut aussi représenter le mois de décembre 2020. Si l'argument test est mis à FALSE, alors aucun test ne sera effectué sur les données en entrée.

Value

En sortie, la fonction retourne une date au même format que l'objet date_ts avec la période inclus entre 1 et la fréquence.

Examples

```
# Formattage inchangée
normalize_date_ts(c(2020L, 1L), frequency_ts = 4L) # 1er trimestre de 2020
normalize_date_ts(c(2020L, 8L), frequency_ts = 12L) # Aout 2020

# Retour dans le passé
normalize_date_ts(c(2020L, 0L), frequency_ts = 4L) # 4ème trimestre de 2019
normalize_date_ts(c(2020L, -10L), frequency_ts = 12L) # février 2019

# Avancée dans le futur
normalize_date_ts(c(2020L, 7L), frequency_ts = 4L) # 3ème trimestre de 2021
normalize_date_ts(c(2020L, 13L), frequency_ts = 4L) # janvier 2021
```

---

previous_date_ts *Obtenir la date précédente*

Description

Obtenir la date précédente

Usage

```
previous_date_ts(date_ts, frequency_ts, lag = 1L)
```

Arguments

date_ts	un vecteur numérique, de préférence integer, au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
frequency_ts	un entier qui vaut 4L (ou 4.0) pour les séries trimestrielles et 12L (ou 12.0) pour les séries mensuelles.
lag	un entier

Details

Lorsqu'on parle de date précédente, on parle de date passée. L'argument lag est entier et désigne le nombre de décalage que l'on affecte à notre date. Par exemple pour des lag positif (1L, 2L, 10L) on désigne le décalage de la période précédente, celle d'avant et celle d'il y a 10 périodes. Cependant, lorsque l'argument lag vaut zéro, la fonction retourne la date inchangée. Aussi lorsque l'argument lag est négatif, la fonction se comporte comme la fonction next_date_ts et retourne les périodes futures et non passées.

Value

En sortie, la fonction retourne un vecteur d'entier qui représente la date à la période passée au format date_ts.

See Also

next_date_ts

Examples

```
previous_date_ts(c(2020L, 4L), frequency_ts = 4L, lag = 2L)
previous_date_ts(c(2021L, 1L), frequency_ts = 4L, lag = -2L)

previous_date_ts(c(2020L, 4L), frequency_ts = 12L, lag = 2L)
previous_date_ts(c(2022L, 6L), frequency_ts = 12L, lag = 12L)
```

---

set_value_ts *Change certaines valeurs d'un ts*

Description

La fonction set_value_ts modifie la ou les valeurs d'un objet ts à une date donnée.

Usage

```
set_value_ts(series, date_ts, replacement)
```

Arguments

series	un objet ts unidimensionnel conforme aux règles de assert_ts
date_ts	un vecteur numérique, de préférence integer, au format AAAA, c(AAAA, MM) ou c(AAAA, TT)
replacement	un vecteur de même type que le ts series

Value

En sortie, la fonction retourne une copie de l'objet series modifié avec les valeurs de replacement imputés à partir de la date date_ts.

Examples

```
set_value_ts(
    series = ev_pib,
    date_ts = c(2021L, 2L),
    replacement = c(1, 2, 3)
)
```

---

substr_year *Retire une année à une date*

Description

La fonction substr_year retire n annnée(s) à une date.

Usage

```
substr_year(date, n = 1L)
```

Arguments

date	un objet de type Date
n	un entier

Value

En sortie, la fonction retourne un objet de type Date (atomic) de longueur 1.

Examples

```
substr_year(as.Date("2000-02-29"), n = 1L)
substr_year(as.Date("2000-02-29"), n = 3L)
substr_year(as.Date("2000-02-29"), n = 4L)
substr_year(as.Date("2000-02-29"), n = 16L)

substr_year(as.Date("2023-01-25"), n = 10L)
substr_year(as.Date("2022-11-01"), n = 3L)
```

`trim2mens` *Conversion entre date mensuelle et trimestrielle*

Description

Les fonctions `trim2mens` et `mens2trim` convertissent une `date_ts` du format mensuelle `c(AAAA, MM)` au format trimestrielle `c(AAAA, TT)`.

Usage

```
trim2mens(date_ts)

mens2trim(date_ts)
```

Arguments

`date_ts`	un vecteur numérique, de préférence `integer`, au format `AAAA`, `c(AAAA, MM)` ou `c(AAAA, TT)`

Value

En sortie, la fonction retourne la date toujours au format `date_ts`.

Examples

```
trim2mens(c(2019L, 4L)) # 4ème trimestre 2019 --> Octobre 2019
trim2mens(c(2020L, 1L)) # 1er trimestre 2020 --> Janvier 2020

mens2trim(c(2019L, 4L)) # Avril 2019 --> 2ème trimestre 2019
mens2trim(c(2020L, 11L)) # Novembre 2020 --> 4ème trimestre 2020
```

ts2df *Convertit un objet* ts *en* data.frame

Description

Convertit un objet ts en data.frame

Usage

```
ts2df(x)
```

Arguments

x	un objet de type ts.

Value

En sortie la fonction retourne un data.frame avec autant de colonnes que x et une de plus pour la date.

Examples

```
ts2df(AirPassengers)
ts2df(Seatbelts)
```

Index

∗ **datasets**
ev_pib, 19

as_yyyymm *(*as_yyyytt*)*, 3
as_yyyytt, 3
assert_date_ts *(*check_date_ts*)*, 4
assert_expression *(*check_expression*)*, 5
assert_frequency *(*check_frequency*)*, 6
assert_scalar_date *(*check_scalar_date*)*, 8
assert_scalar_integer *(*check_scalar_integer*)*, 9
assert_scalar_integer(), *11*
assert_scalar_natural *(*check_scalar_natural*)*, 10
assert_scalar_natural(), *10*
assert_timeunits *(*check_timeunits*)*, 12
assert_ts *(*check_ts*)*, 13

check_date_ts, 4
check_expression, 5
check_frequency, 6
check_scalar_date, 8
check_scalar_integer, 9
check_scalar_integer(), *11*
check_scalar_natural, 10
check_scalar_natural(), *10*
check_timeunits, 12
check_ts, 13
combine2ts, 14

date2date_ts, 15
date_ts2date, 16
date_ts2timeunits, 17
diff_periode, 18

ev_pib, 19
extend_ts, 19

first_date, 21

get_value_ts, 22

is_before, 23

last_date, 24
libelles, 25

mens2trim *(*trim2mens*)*, 31

na_trim, 26
next_date_ts, 26
normalize_date_ts, 27

previous_date_ts, 28

set_value_ts, 29
substr_year, 30

trim2mens, 31
ts2df, 32