Package 'AirMonitor'

August 25, 2023

Type Package

Version 0.3.12

Title Air Quality Data Analysis

Maintainer Jonathan Callahan <jonathan.s.callahan@gmail.com>

Description Utilities for working with hourly air quality monitoring data with a focus on small particulates (PM2.5). A compact data model is structured as a list with two dataframes. A 'meta' dataframe contains spatial and measuring device metadata associated with deployments at known locations. A 'data' dataframe contains a 'datetime' column followed by columns of measurements associated with each ``device-deployment''. Algorithms to calculate NowCast and the associated Air Quality Index (AQI) are defined at the US Environmental Projection Agency AirNow program: <https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf>.

License GPL-3

URL https://github.com/MazamaScience/AirMonitor

BugReports https://github.com/MazamaScience/AirMonitor/issues

Depends R (>= 4.0.0)

Imports dplyr, dygraphs, leaflet, lubridate, magrittr, MazamaCoreUtils (>= 0.4.13), MazamaRollUtils (>= 0.1.3), MazamaTimeSeries (>= 0.2.12), readr, rlang (>= 1.0.0), stringr, tidyselect, xts

Suggests knitr, markdown, testthat, rmarkdown, roxygen2

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 7.2.3

NeedsCompilation no

Author Jonathan Callahan [aut, cre],
Spencer Pease [ctb],
Hans Martin [ctb],
Rex Thompson [ctb]

Repository CRAN

Date/Publication 2023-08-25 21:40:07 UTC

R topics documented:

`addAQILegend` *Add an AQI legend to a map*

Description

This function is a convenience wrapper around `graphics::legend()`. It will show the AQI colors and names by default if `col` and `legend` are not specified.

AQI categories are arranged with lower levels at the bottom of the legend to match the arrangement in the plot. This is different from the default "reading order" so you may wish to reverse the order of user supplied arguments with `rev()` .

Usage

```
addAQILegend(
  x = "topright",
  y = NULL,
  pollutant = c("PM2.5", "CO", "OZONE", "PM10", "AQI"),
  palette = c("EPA", "subdued", "deuteranopia"),
  languageCode = c("eng", "spa"),
  ...
)
```

Arguments

x	x Coordinate passed on to the `legend()` command.
y	y Coordinate passed on to the `legend()` command.
pollutant	EPA AQS criteria pollutant.
palette	Named color palette to use for AQI categories.
languageCode	ISO 639-2 alpha-3 language code.
...	Additional arguments to be passed to `legend()`.

Value

A list with components `rect` and `text` is returned invisbly. (See legend.)

---

addAQILines *Add AQI lines to a plot*

Description

Draws AQI lines across a plot at the levels appropriate for The monitor_timeseriesPlot function uses this function internally when specifying `addAQI = TRUE`. `pollutant`.

Usage

```
addAQILines(
  pollutant = c("PM2.5", "CO", "OZONE", "PM10", "AQI"),
  palette = c("EPA", "subdued", "deuteranopia"),
  ...
)
```

Arguments

pollutant	EPA AQS criteria pollutant.
palette	Named color palette to use for AQI categories.
...	additional arguments to be passed to `abline()`

Value

No return value, called to add lines to a time series plot.

addAQIStackedBar *Create stacked AQI bar*

Description

Draws a stacked bar indicating AQI levels on one side of a plot The monitor_timeseriesPlot function uses this function internally when specifying `addAQI = TRUE`.

Usage

```
addAQIStackedBar(
  pollutant = c("PM2.5", "CO", "OZONE", "PM10", "AQI"),
  palette = c("EPA", "subdued", "deuteranopia"),
  width = 0.01,
  height = 1,
  pos = c("left", "right")
)
```

Arguments

`pollutant`	EPA AQS criteria pollutant.
`palette`	Named color palette to use for AQI categories.
`width`	Width of the bar as a fraction of the width of the plot area.
`height`	Height of the bar as a fraction of the height of the plot area.
`pos`	Position of the stacked bar relative to the plot.

Value

No return value, called to add color bars to a time series plot.

addShadedNight *Add nighttime shading to a timeseries plot*

Description

Draw shading rectangles on a plot to indicate nighttime hours. The monitor_timeseriesPlot function uses this function internally when specifying `shadedNight = TRUE`.

Usage

```
addShadedNight(timeInfo, col = adjustcolor("black", 0.1))
```

Arguments

timeInfo	dataframe as returned by MazamaTimeSeries::monitor_timeInfo()
col	Color used to shade nights.

Value

No return value, called to add day/night shading to a timeseries plot.

AirFire_S3_archiveBaseUrl *USFS maintained archive base URL*

Description

The US Forest Service AirFire group maintains an archive of processed monitoring data. The base URL for this archive is used as the default in all ~_load() functions.

"https://airfire-data-exports.s3.us-west-2.amazonaws.com/monitoring/v2"

Usage

```
AirFire_S3_archiveBaseUrl
```

Format

A url

Details

AirFire_S3_archiveBaseUrl

AirMonitor *Air Quality Data Analysis*

Description

Utilities for working with hourly air quality monitoring data with a focus on small particulates (PM2.5). A compact data model is structured as a list with two dataframes. A 'meta' dataframe contains spatial and measuring device metadata associated with deployments at known locations. A 'data' dataframe contains a 'datetime' column followed by columns of measurements associated with each "device-deployment".

`airnow_loadAnnual` *Load annual AirNow monitoring data*

Description

Loads pre-generated .rda files containing hourly AirNow data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function contain a single year's worth of data

For the most recent data in the last 10 days, use `airnow_loadLatest()`.

For daily updates covering the most recent 45 days, use `airnow_loadDaily()`.

For archival data for a specific month, use `airnow_loadMonthly()`.

Pre-processed AirNow exists for the following parameters:

1. PM2.5

Usage

```
airnow_loadAnnual(
  year = NULL,
  archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  parameterName = "PM2.5"
)
```

Arguments

`year`	Year [YYYY].
`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`parameterName`	One of the EPA AQS criteria parameter names.

Value

A *mts_monitor* object with AirNow data. (A list with `meta` and `data` dataframes.)

See Also

`airnow_loadDaily`

`airnow_loadLatest`

`airnow_loadMonthly`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

# See https://en.wikipedia.org/wiki/2017_Montana_wildfires

# Daily Barplot of Montana wildfires
airnow_loadAnnual(2017) \
  monitor_filter(stateCode == "MT") \
  monitor_filterDate(20170701, 20170930, timezone = "America/Denver") \
  monitor_dailyStatistic() \
  monitor_timeseriesPlot(
    ylim = c(0, 300),
    xpd = NA,
    addAQI = TRUE,
    main = "Montana 2017 -- AirNow Daily Average PM2.5"
  )

}, silent = FALSE)

## End(Not run)
```

`airnow_loadDaily` *Load daily AirNow monitoring data*

Description

Loads pre-generated .rda files containing hourly AirNow data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated once per day and contain data for the previous 45 days.

For the most recent data in the last 10 days, use `airnow_loadLatest()`.

For data extended more than 45 days into the past, use `airnow_loadAnnual()`.

Pre-processed AirNow exists for the following parameters:

1. PM2.5
2. PM2.5_nowcast

Usage

```
airnow_loadDaily(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
 archiveBaseDir = NULL,
```

```
  QC_negativeValues = c("zero", "na", "ignore"),
  parameterName = "PM2.5"
)
```

Arguments

archiveBaseUrl	Base URL for monitoring v2 data files.
archiveBaseDir	Local base directory for monitoring v2 data files.
QC_negativeValues	Type of QC to apply to negative values.
parameterName	One of the EPA AQS criteria parameter names.

Value

A *mts_monitor* object with AirNow data. (A list with meta and data dataframes.)

See Also

airnow_loadAnnual

airnow_loadLatest

airnow_loadMonthly

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

airnow_loadDaily() \
  monitor_filter(stateCode == "WA") \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

---

airnow_loadLatest *Load most recent AirNow monitoring data*

Description

Loads pre-generated .rda files containing the most recent AirNow data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated multiple times an hour and contain data for the previous 10 days.

For daily updates covering the most recent 45 days, use `airnow_loadDaily()`.

For data extended more than 45 days into the past, use `airnow_loadAnnual()`.

Pre-processed AirNow exists for the following parameters:

1. PM2.5
2. PM2.5_nowcast

Usage

```
airnow_loadLatest(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
 archiveBaseDir = NULL,
 QC_negativeValues = c("zero", "na", "ignore"),
 parameterName = "PM2.5"
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`parameterName`	One of the EPA AQS criteria parameter names.

Value

A *mts_monitor* object with AirNow data. (A list with `meta` and `data` dataframes.)

See Also

`airnow_loadAnnual`

`airnow_loadDaily`

`airnow_loadMonthly`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

airnow_loadLatest() \
  monitor_filter(stateCode == "WA") \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`airnow_loadMonthly` *Load monthly AirNow monitoring data*

Description

Loads pre-generated .rda files containing hourly AirNow data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function contain a single month's worth of data

For the most recent data in the last 10 days, use `airnow_loadLatest()`.

For daily updates covering the most recent 45 days, use `airnow_loadDaily()`.

For data extended more than 45 days into the past, use `airnow_loadAnnual()`.

Pre-processed AirNow exists for the following parameters:

1. PM2.5

Usage

```
airnow_loadMonthly(
  monthStamp = NULL,
  archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  parameterName = "PM2.5"
)
```

Arguments

monthStamp	Year-month [YYYYmm].
archiveBaseUrl	Base URL for monitoring v2 data files.
archiveBaseDir	Local base directory for monitoring v2 data files.
QC_negativeValues	Type of QC to apply to negative values.
parameterName	One of the EPA AQS criteria parameter names.

Value

A *mts_monitor* object with AirNow data. (A list with `meta` and `data` dataframes.)

---

airsis_loadAnnual *Load annual AIRSIS monitoring data*

Description

Loads pre-generated .rda files containing annual AIRSIS data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

Current year files loaded by this function are updated once per week.

For the most recent data in the last 10 days, use `airsis_loadLatest()`.

For daily updates covering the most recent 45 days, use `airsis_loadDaily()`.

Usage

```
airsis_loadAnnual(
  year = NULL,
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  QC_removeSuspectData = TRUE
)
```

Arguments

year	Year [YYYY].
archiveBaseUrl	Base URL for monitoring v2 data files.
archiveBaseDir	Local base directory for monitoring v2 data files.
QC_negativeValues	Type of QC to apply to negative values.
QC_removeSuspectData	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with AIRSIS data. (A list with `meta` and `data` dataframes.)

Note

Some older AIRSIS timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`airsis_loadDaily`

`airsis_loadLatest`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

# See https://en.wikipedia.org/wiki/Camp_Fire_(2018)

# AIRSIS monitors during the Camp Fire
airsis_loadAnnual(2018) \
  monitor_filter(stateCode == "CA") \
  monitor_filterDate(20181101, 20181201) \
  monitor_dropEmpty() \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

---

airsis_loadDaily *Load daily AIRSIS monitoring data*

Description

Loads pre-generated .rda files containing daily AIRSIS data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated once per day and contain data for the previous 45 days.

For the most recent data in the last 10 days, use `airsis_loadLatest()`.

For data extended more than 45 days into the past, use `airsis_loadAnnual()`.

Usage

```
airsis_loadDaily(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
 archiveBaseDir = NULL,
 QC_negativeValues = c("zero", "na", "ignore"),
 QC_removeSuspectData = TRUE
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`QC_removeSuspectData`	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with AIRSIS data. (A list with `meta` and `data` dataframes.)

Note

Some older AIRSIS timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`airsis_loadAnnual`

`airsis_loadLatest`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

airsis_loadDaily()\ %>\
```

```
  monitor_filter(stateCode == "CA") \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`airsis_loadLatest` *Load most recent AIRSIS monitoring data*

Description

Loads pre-generated .rda files containing the most recent AIRSIS data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated multiple times an hour and contain data for the previous 10 days.

For daily updates covering the most recent 45 days, use `airsis_loadDaily()`.

For data extended more than 45 days into the past, use `airsis_loadAnnual()`.

Usage

```
airsis_loadLatest(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
 archiveBaseDir = NULL,
 QC_negativeValues = c("zero", "na", "ignore"),
 QC_removeSuspectData = TRUE
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`QC_removeSuspectData`	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with AIRSIS data. (A list with `meta` and `data` dataframes.)

Note

Some older AIRSIS timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`airsis_loadAnnual`

`airsis_loadDaily`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

airsis_loadLatest()\ %>\
  monitor_filter(stateCode == "CA") \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`aqiColors` *Generate AQI colors*

Description

This function uses the `leaflet::colorBin()` function to return a vector or matrix of colors derived from data values.

Usage

```
aqiColors(
  x,
  pollutant = c("PM2.5", "AQI", "CO", "NO", "OZONE", "PM10", "SO2"),
  palette = c("EPA", "subdued", "deuteranopia"),
  na.color = NA
)
```

Arguments

x	Vector or matrix of PM2.5 values or an *mts_monitor* object.
pollutant	EPA AQS criteria pollutant.
palette	Named color palette to use for AQI categories.
na.color	Color assigned to missing values.

Value

A vector or matrix of AQI colors to be used in maps and plots.

Examples

```
library(AirMonitor)

# Fancy plot based on pm2.5 values
pm2.5 <- Carmel_Valley$data[,2]
Carmel_Valley %>%
  monitor_timeseriesPlot(
    shadedNight = TRUE,
    pch = 16,
    cex = pmax(pm2.5 / 100, 0.5),
    col = aqiColors(pm2.5),
    opacity = 0.8
  )
```

Camp_Fire *Camp Fire example dataset*

Description

The Camp_Fire dataset provides a quickly loadable version of a *mts_monitor* object for practicing and code examples.

Usage

```
Camp_Fire
```

Format

A *mts_monitor* object with 360 rows and 134 columns of data.

Details

The 2018 Camp Fire was the deadliest and most destructive wildfire in California's history, and the most expensive natural disaster in the world in 2018 in terms of insured losses. The fire caused at least 85 civilian fatalities and injured 12 civilians and five firefighters. It covered an area of 153,336 acres and destroyed more than 18,000 structures, most with the first 4 hours. Smoke from the fire resulted in the worst air pollution ever for the San Francisco Bay Area and Sacramento Valley.

This dataset was was generated on 2022-10-12 by running:

```
library(AirMonitor)

Camp_Fire <-
  monitor_loadAnnual(2018) %>%
  monitor_filter(stateCode == 'CA') %>%
  monitor_filterDate(
    startdate = 20181108,
    enddate = 20181123,
    timezone = "America/Los_Angeles"
 ) %>%
 monitor_dropEmpty()

save(Camp_Fire, file = "data/Camp_Fire.rda")
```

`Carmel_Valley` *Carmel Valley example dataset*

Description

The `Carmel_Valley` dataset provides a quickly loadable version of a *mts_monitor* object for practicing and code examples.

Usage

```
Carmel_Valley
```

Format

A *mts_monitor* object with 576 rows and 2 columns of data.

Details

In August of 2016, the Soberanes fire in California burned along the Big Sur coast. At the time, it was the most expensive wildfire in US history. This dataset contains PM2.5 monitoring data for the monitor in Carmel Valley which shows heavy smoke as well as strong diurnal cycles associated with sea breezes. Data are stored as a *mts_monitor* object and are used in some examples in the package documentation.

This dataset was generated on 2022-10-12 by running:

```
library(AirMonitor)

Carmel_Valley <-
  airnow_loadAnnual(2016) %>%
  monitor_filterMeta(deviceDeploymentID == "a9572a904a4ed46d_840060530002") %>%
  monitor_filterDate(20160722, 20160815)

save(Carmel_Valley, file = "data/Carmel_Valley.rda")
```

CONUS *CONUS state codes*

Description

State codes for the 48 contiguous states +DC that make up the CONtinental US.

```
CONUS <- c( "AL","AZ","AR","CA","CO","CT","DE","FL","GA", "ID","IL","IN","IA","KS","KY","LA","ME","MD
"MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ", "NM","NY","NC","ND","OH","OK","OR","PA","RI","SC"
"SD","TN","TX","UT","VT","VA","WA","WV","WI","WY", "DC" )
```

Usage

```
CONUS
```

Format

A vector with 49 elements

Details

CONUS state codes

---

coreMetadataNames *Names of standard metadata columns*

Description

Vector of names of the required `monitor$meta` columns. These represent metadata columns that must exist in every valid *mts_monitor* object. Any number of additional columns may also be present.

Usage

```
coreMetadataNames
```

Format

A vector of character strings

Details

coreMetadataNames

Examples

```
print(coreMetadataNames, width = 80)
```

---

epa_aqs_loadAnnual *Load annual AirNow monitoring data*

Description

Loads pre-generated .rda files containing hourly AirNow data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function contain a single year's worth of data.

Pre-processed AirNow exists for the following parameter codes:

1. 88101 – PM2.5 FRM/FEM Mass
2. 88502 – PM2.5 non FRM/FEM Mass

Specifying `parameterCode = "PM2.5"` will merge records from both sources.

Usage

```
epa_aqs_loadAnnual(
  year = NULL,
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  parameterCode = c("PM2.5", "88101", "88502")
)
```

Arguments

`year`	Year [YYYY].
`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`parameterCode`	One of the EPA AQS criteria parameter codes.

Value

A *mts_monitor* object with EPA AQS data. (A list with `meta` and `data` dataframes.)

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

# See https://en.wikipedia.org/wiki/2017_Montana_wildfires

# Daily Barplot of Montana wildfires
epa_aqs_loadAnnual(2015) \
  monitor_filter(stateCode == "WA") \
  monitor_filterDate(20150724, 20150907) \
  monitor_dailyStatistic() \
  monitor_timeseriesPlot(
    main = "Washington 2015 -- AirNow Daily Average PM2.5"
  )

}, silent = FALSE)

## End(Not run)
```

---

monitor_aqi *Calculate hourly NowCast-based AQI values*

Description

Nowcast and AQI algorithms are applied to the data in the monitor object. A modified `mts_monitor` object is returned whre values have been replaced with their Air Quality Index equivalents. See monitor_nowcast.

Usage

```
monitor_aqi(
  monitor,
  version = c("pm", "pmAsian", "ozone"),
  includeShortTerm = FALSE
)
```

Arguments

`monitor`	*mts_monitor* object.
`version`	Name of the type of nowcast algorithm to be used.
`includeShortTerm`	Logical specifying whether to alcluate preliminary NowCast values starting with the 2nd hour.

Value

A modified mts_monitor object containing AQI values. (A list with meta and data dataframes.)

References

https://en.wikipedia.org/wiki/Nowcast_(Air_Quality_Index)

https://www.airnow.gov/aqi/aqi-basics/

---

monitor_arrange *Order* mts_monitor *time series by metadata values*

Description

The variable(s) in ... are used to specify columns of monitor$meta to use for ordering. Under the hood, this function uses arrange on monitor$meta and then reorders monitor$data to match.

Usage

```
monitor_arrange(monitor, ...)
```

Arguments

monitor	*mts_monitor* object.
...	variables in mts$meta.

Value

A reorderd version of the incoming *mts* time series object. (A list with meta and data dataframes.)

See Also

monitor_select

Examples

```
library(AirMonitor)

Camp_Fire$meta$elevation[1:10]

byElevation <-
  Camp_Fire %>%
  monitor_arrange(elevation)

byElevation$meta$elevation[1:10]
```

monitor_bestTimezone *Return the most common timezone*

Description

Evaluates all timezones in `monitor` and returns the most common one. In the case of a tie, the alphabetically first one is returned.

Usage

```
monitor_bestTimezone(monitor = NULL)
```

Arguments

`monitor`	*mts_monitor* object.

Value

A valid `base::OlsonNames()` timezone.

monitor_check *Check an* mts_monitor *object for validity.*

Description

Checks on the validity of an *mts_monitor* object. If any test fails, this function will stop with a warning message.

Usage

```
monitor_check(monitor)
```

Arguments

`monitor`	*mts_monitor* object.

Value

Invisibly returns `TRUE` if `mts_monitor` has the correct structure. Stops with an error message otherwise.

monitor_collapse *Collapse an* mts_monitor *object into a single time series*

Description

Collapses data from all time series in a mts_monitor into a single-time series *mts_monitor* object using the function provided in the FUN argument. The single-time series result will be located at the mean longitude and latitude unless longitude and latitude parameters are specified.

Any columns of monitor$meta that are constant across all records will be retained in the returned *mts_monitor* meta dataframe.

The core metadata associated with this location (*e.g.* countryCode, stateCode, timezone, ...) will be determined from the most common (or average) value found in monitor$meta. This will be a reasonable assumption for the vast majority of intended use cases where data from multiple instruments in close proximity are averaged together.

Usage

```
monitor_collapse(
  monitor,
  longitude = NULL,
  latitude = NULL,
  deviceID = "generatedID",
  FUN = mean,
  na.rm = TRUE,
  ...
)
```

Arguments

monitor	*mts_monitor* object.
longitude	Longitude of the collapsed time series.
latitude	Latitude of the collapsed time series.
deviceID	Device identifier for the collapsed time series.
FUN	Function used to collapse multiple time series.
na.rm	Logical specifying whether NA values should be ignored when FUN is applied.
...	additional arguments to be passed on to the apply() function.

Value

A *mts_monitor* object representing a single time series. (A list with meta and data dataframes.)

Note

After FUN is applied, values of +/-Inf and NaN are converted to NA. This is a convenience for the common case where FUN = min/max or FUN = mean and some of the time steps have all missing values. See the R documentation for min for an explanation.

Examples

```
library(AirMonitor)

# Lane County, Oregon AQSIDs all begin with "41039"
LaneCounty <-
  NW_Megafires %>%
  monitor_filter(stringr::str_detect(AQSID, '^41039')) %>%
  monitor_filterDate(20150821, 20150828)

# Get min/max for all monitors
LaneCounty_min <- monitor_collapse(LaneCounty, deviceID = 'LaneCounty_min', FUN = min)
LaneCounty_max <- monitor_collapse(LaneCounty, deviceID = 'LaneCounty_max', FUN = max)

# Create plot
monitor_timeseriesPlot(
  LaneCounty,
  shadedNight = TRUE,
  main = "Lane County Range of PM2.5 Values"
)

# Add min/max lines
monitor_timeseriesPlot(LaneCounty_max, col = 'red', type = 's', add = TRUE)
monitor_timeseriesPlot(LaneCounty_min, col = 'blue', type = 's', add = TRUE)
```

---

monitor_combine *Combine multiple* mts_monitor *objects*

Description

Create a combined *mts_monitor* from any number of *mts_monitor* objects or from a list of *mts_monitor* objects. The resulting *mts_monitor* object with contain all `deviceDeploymentIDs` found in any incoming *mts_monitor* and will have a regular time axis covering the the entire range of incoming data.

If incoming time ranges are tempporally non-contiguous, the resulting *mts_monitor* will have gaps filled with `NA` values.

An error is generated if the incoming *mts_monitor* objects have non-identical metadata for the same `deviceDeploymentID` unless `replaceMeta = TRUE`.

Usage

```
monitor_combine(
  ...,
  replaceMeta = FALSE,
  overlapStrategy = c("replace all", "replace na")
)
```

Arguments

...	Any number of valid emphmts_monitor objects or a list of objects.
replaceMeta	Logical specifying whether to allow replacement of metadata associated when duplicate deviceDeploymentIDs are encountered.
overlapStrategy	Strategy to use when data found in time series overlaps.

Value

A combined mts_monitor object. (A list with meta and data dataframes.)

Note

Data are combined with a "later is better" sensibility where any data overlaps exist. Incoming *mts_monitor* objects are ordered based on the time stamp of their last record. Any data records found in a "later" *mts_monitor* will overwrite data associated with an "earlier" *mts_monitor*.

With overlapStrategy = "replace all", any data records found in "later" *mts_monitor* objects are preferentially retained before the "shared" data are finally reordered by ascending datetime.

With overlapStrategy = "replace missing", only missing values in "earlier" *mts_monitor* objects are replaced with data records from "later" time series.

Examples

```
library(AirMonitor)

# Two monitors near Pendelton, Oregon
#
# Use the interactive map to get the deviceDeploymentIDs
#   NW_Megafires %>% monitor_leaflet()

Pendleton_West <-
  NW_Megafires %>%
  monitor_select("f187226671d1109a_410590121_03") %>%
  monitor_filterDatetime(2015082300, 2015082305)

Pendleton_East <-
  NW_Megafires %>%
  monitor_select("6c906c6d1cf46b53_410597002_02") %>%
  monitor_filterDatetime(2015082300, 2015082305)

monitor_combine(Pendleton_West, Pendleton_East) %>%
  monitor_getData()
```

`monitor_dailyBarplot` *Create daily barplot*

Description

Creates a daily barplot of data from a *mts_monitor* object.

Reasonable defaults are chosen for annotations and plot characteristics. Users can override any defaults by passing in parameters accepted by `graphics::barplot`.

Usage

```
monitor_dailyBarplot(
  monitor = NULL,
  id = NULL,
  add = FALSE,
  addAQI = FALSE,
  palette = c("EPA", "subdued", "deuteranopia"),
  opacity = NULL,
  ...,
  minHours = 18,
  dayBoundary = c("clock", "LST")
)
```

Arguments

`monitor`	*mts_monitor* object.
`id`	`deviceDeploymentID` for a single time series found in `monitor`. (Optional if `monitor` contains only a single time series.)
`add`	Logical specifying whether to add to the current plot.
`addAQI`	Logical specifying whether to add visual AQI decorations.
`palette`	Named color palette to use when adding AQI decorations.
`opacity`	Opacity to use for bars.
`...`	Additional arguments to be passed to `graphics::barplot()`.
`minHours`	Minimum number of valid hourly records per day required to calculate statistics. Days with fewer valid records will be assigned `NA`.
`dayBoundary`	Treatment of daylight savings time: "clock" uses daylight savings time as defined in the local timezone, "LST" uses "local standard time" all year round.

Value

No return value. This function is called to draw an air quality daily average plot on the active graphics device.

Note

The underlying axis for this plot is not a time axis so you cannot use this function to "add" bars on top of a `monitor_timeseriesPlot()`. See the **AirMonitorPlots** package for more flexibility in plotting.

Examples

```
library(AirMonitor)

Carmel_Valley %>%
  monitor_dailyBarplot()
```

`monitor_dailyStatistic` *Create daily statistics for each monitor in an* mts_monitor *object*

Description

Daily statstics are calculated for each time series in `monitor$data` using `FUN` and any arguments passed in `....`

Because the returned *mts_monitor* object is defined on a daily axis in a specific time zone, it is important that the incoming `monitor` contain timeseries associated with a single time zone.

Usage

```
monitor_dailyStatistic(
  monitor = NULL,
  FUN = mean,
  na.rm = TRUE,
  ...,
  minHours = 18,
  dayBoundary = c("clock", "LST")
)
```

Arguments

`monitor`	*mts_monitor* object.
`FUN`	Function used to create daily statistics.
`na.rm`	Value passed on to `FUN`. If `FUN` does not use `na.rm`, this should be set to `NULL`.
`...`	Additional arguments to be passed to `FUN`.
`minHours`	Minimum number of valid hourly records per day required to calculate statistics. Days with fewer valid records will be assigned `NA`.
`dayBoundary`	Treatment of daylight savings time: "clock" uses daylight savings time as defined in the local timezone, "LST" uses "local standard time" all year round.

Value

A *mts_monitor* object containing daily statistical summaries. (A list with `meta` and `data` dataframes.)

Note

When `dayBoundary = "clock"`, the returned `monitor$data$datetime` time axis will be defined in the local timezone (not "UTC") with days defined by midnight as it appears on a clock in that timezone. The transition from DST to standard time will result in a 23 hour day and standard to DST in a 25 hour day.

When `dayBoundary = "LST"`, the returned `monitor$data$datetime` time axis will be defined in "UTC" with times as they *appear* in standard time in the local timezone. These days will be one hour off from clock time during DST but every day will consist of 24 hours.

Examples

```
library(AirMonitor)

Carmel_Valley %>%
  monitor_dailyStatistic(max) %>%
  monitor_getData()

Carmel_Valley %>%
  monitor_dailyStatistic(min) %>%
  monitor_getData()
```

---

monitor_dailyThreshold

Daily counts of values at or above a threshold

Description

Calculates the number of hours per day each time series in `monitor` was at or above a given threshold.

Because the returned *mts_monitor* object is defined on a daily axis in a specific time zone, it is important that the incoming `monitor` contain only timeseries within a single time zone.

Usage

```
monitor_dailyThreshold(
  monitor = NULL,
  threshold = NULL,
  na.rm = TRUE,
  minHours = 18,
  dayBoundary = c("clock", "LST")
)
```

Arguments

monitor	*mts_monitor* object.
threshold	AQI level name (e.g. "unhealthy") or numerical threshold at and above which a measurement is counted.
na.rm	Logical value indicating whether NA values should be ignored.
minHours	Minimum number of valid hourly records per day required to calculate statistics. Days with fewer valid records will be assigned NA.
dayBoundary	Treatment of daylight savings time: "clock" uses daylight savings time as defined in the local timezone, "LST" uses "local standard time" all year round.

Value

A *mts_monitor* object containing daily counts of hours at or above a threshold value. (A list with meta and data dataframes.)

Note

When dayBoundary = "clock", the returned monitor$data$datetime time axis will be defined in the local timezone (not "UTC") with days defined by midnight as it appears on a clock in that timezone. The transition from DST to standard time will result in a 23 hour day and standard to DST in a 25 hour day.

When dayBoundary = "LST", the returned monitor$data$datetime time axis will be defined in "UTC" with times as they *appear* in standard time in the local timezone. These days will be one hour off from clock time during DST but every day will consist of 24 hours.

Examples

```
library(AirMonitor)

Carmel_Valley %>%
  monitor_dailyThreshold("Moderate") %>%
  monitor_getData()

Carmel_Valley %>%
  monitor_dailyThreshold("Unhealthy") %>%
  monitor_getData()
```

---

monitor_distinct *Retain only distinct data records in monitor$data*

Description

Two successive steps are used to guarantee that the datetime axis contains no repeated values:

1. remove any duplicate records
2. guarantee that rows are in datetime order

Usage

```
monitor_distinct(monitor)
```

Arguments

monitor	*mts_monitor* object

Value

A *mts_monitor* object with no duplicated data records. (A list with `meta` and `data` dataframes.)

Note

This function is primarily for package-internal use.

---

monitor_dropEmpty *Drop device deployments with all missing data*

Description

The incoming *mts_monitor* object is subset to retain only time series with valid data.

Usage

```
monitor_dropEmpty(monitor)
```

Arguments

monitor	*mts_monitor* object. (A list with `meta` and `data` dataframes.)

Value

A subset of the incoming `mts_monitor`. (A list with `meta` and `data` dataframes.)

`monitor_dygraph` *Create Interactive Time Series Plot*

Description

This function creates interactive graphs that will be displayed in RStudio's 'Viewer' tab.

Usage

```
monitor_dygraph(
  monitor,
  title = "title",
  ylab = "PM2.5 Concentration",
  rollPeriod = 1,
  showLegend = TRUE
)
```

Arguments

`monitor`	*mts_monitor* object.
`title`	Title text.
`ylab`	Title for the y axis
`rollPeriod`	Rolling mean to be applied to the data.
`showLegend`	Logical to toggle display of the legend.

Value

Initiates the interactive dygraph plot in RStudio's 'Viewer' tab.

Examples

```
## Not run:
library(AirMonitor)

# Multiple monitors
Camp_Fire %>%
  monitor_filter(countyName == "Alameda") %>%
  monitor_dygraph()

## End(Not run)
```

`monitor_filterByDistance` *Filter by distance from a target location*

Description

Filters the `monitor` argument to include only those time series located within a certain radius of a target location. If no time series fall within the specified `radius`, an empty *mts_monitor* object will be returned.

When `count` is used, a *mts_monitor* object is created containing **up to** `count` time series, ordered by increasing distance from the target location. Note that the number of monitors returned may be less than the specified `count` value if fewer than `count` time series are found within the target area.

Usage

```
monitor_filterByDistance(
  monitor,
  longitude = NULL,
  latitude = NULL,
  radius = 50,
  count = NULL,
  addToMeta = FALSE
)
```

Arguments

`monitor`	*mts_monitor* object.
`longitude`	Target longitude.
`latitude`	Target.
`radius`	Distance (m) of radius defining a target area.
`count`	Number of time series to return.
`addToMeta`	Logical specifying whether to add `distanceFromTarget` as a field in `monitor$meta`.

Value

A *mts_monitor* object with monitors near a location.

Note

The returned *mts_monitor* will have an extra `distance`. (A list with `meta` and `data` dataframes.)

Examples

```
library(AirMonitor)

# Walla Walla
longitude <- -118.330278
latitude <- 46.065

Walla_Walla_monitors <-
  NW_Megafires %>%
  monitor_filterByDistance(
    longitude = -118.330,
    latitude = 46.065,
    radius = 50000,     # 50 km
    addToMeta = TRUE
  )

Walla_Walla_monitors %>%
  monitor_getMeta() %>%
  dplyr::select(c("locationName", "distanceFromTarget"))
```

`monitor_filterDate` *Date filtering for* mts_monitor *objects*

Description

Subsets a *mts_monitor* object by date. This function always filters to day-boundaries. For sub-day filtering, use `monitor_filterDatetime()`.

Dates can be anything that is understood by `MazamaCoreUtils::parseDatetime()` including either of the following recommended formats:

- `"YYYYmmdd"`
- `"YYYY-mm-dd"`

If either `startdate` or `enddate` is not provided, the start/end of the *mts_monitor* time axis will be used.

Timezone determination precedence assumes that if you are passing in `POSIXct` values then you know what you are doing.

1. get timezone from `startdate` if it is `POSIXct`
2. use passed in `timezone`
3. get timezone from `mts_monitor`

Usage

```
monitor_filterDate(
  monitor = NULL,
  startdate = NULL,
  enddate = NULL,
  timezone = NULL,
  unit = "sec",
  ceilingStart = FALSE,
  ceilingEnd = FALSE
)
```

Arguments

monitor	*mts_monitor* object.
startdate	Desired start datetime (ISO 8601).
enddate	Desired end datetime (ISO 8601).
timezone	Olson timezone used to interpret dates.
unit	Units used to determine time at end-of-day.
ceilingStart	Logical instruction to apply ceiling_date to the startdate rather than floor_date
ceilingEnd	Logical instruction to apply ceiling_date to the enddate rather than floor_date

Value

A subset of the given *mts_monitor* object. (A list with meta and data dataframes.)

Note

The returned data will run from the beginning of startdate until the **beginning** of enddate – *i.e.* no values associated with enddate will be returned. The exception being when enddate is less than 24 hours after startdate. In that case, a single day is returned.

See Also

monitor_filterDatetime

monitor_filterMeta

Examples

```
library(AirMonitor)

Camp_Fire %>%
  monitor_timeRange()

# Day boundaries returned in "UTC"
Camp_Fire %>%
  monitor_filterDate(
    "2018-11-15",
    "2018-11-22",
```

```
  timezone = "America/Los_Angeles"
) %>%
monitor_timeRange()

# Day boundaries returned in "America/Los_Angeles"
Camp_Fire %>%
  monitor_filterDatetime(
    "20181115",
    "20181122",
    timezone = "America/Los_Angeles"
  ) %>%
  monitor_timeRange(
    timezone = "America/Los_Angeles"
  )
```

`monitor_filterDatetime`

Datetime filtering for `mts_monitor` *objects*

Description

Subsets a *mts_monitor* object by datetime. This function allows for sub-day filtering as opposed to `monitor_filterDate()` which always filters to day-boundaries.

Datetimes can be anything that is understood by `MazamaCoreUtils::parseDatetime()`. For non-`POSIXct` values, the recommended format is `"YYYY-mm-dd HH:MM:SS"`.

If either `startdate` or `enddate` is not provided, the start/end of the *mts_monitor* time axis will be used.

Timezone determination precedence assumes that if you are passing in `POSIXct` values then you know what you are doing.

1. get timezone from `startdate` if it is `POSIXct`
2. use passed in `timezone`
3. get timezone from `mts_monitor`

Usage

```
monitor_filterDatetime(
  monitor = NULL,
  startdate = NULL,
  enddate = NULL,
  timezone = NULL,
  unit = "sec",
  ceilingStart = FALSE,
  ceilingEnd = FALSE
)
```

Arguments

monitor	*mts_monitor* object.
startdate	Desired start datetime (ISO 8601).
enddate	Desired end datetime (ISO 8601).
timezone	Olson timezone used to interpret startdate and enddate.
unit	Units used to determine time at end-of-day.
ceilingStart	Logical specifying application of ceiling_date to the startdate rather than floor_date
ceilingEnd	Logical specifying application of ceiling_date to the enddate rather than floor_date

Value

A subset of the given *mts_monitor* object. (A list with meta and data dataframes.)

See Also

monitor_filterDate

monitor_filterMeta

Examples

```
library(AirMonitor)

Camp_Fire %>%
  monitor_timeRange()

# Reduced time range returned in "UTC"
Camp_Fire %>%
  monitor_filterDatetime(
    "2018-11-15 02:00:00",
    "2018-11-22 06:00:00",
    timezone = "America/Los_Angeles"
  ) %>%
  monitor_timeRange()

# Reduced time range returned in "America/Los_Angeles"
Camp_Fire %>%
  monitor_filterDatetime(
    "2018111502",
    "2018112206",
    timezone = "America/Los_Angeles"
  ) %>%
  monitor_timeRange(
    timezone = "America/Los_Angeles"
  )
```

monitor_filterMeta *General purpose metadata filtering for* mts_monitor *objects*

Description

A generalized metadata filter for *mts_monitor* objects to choose cases where conditions are true. Multiple conditions are combined with & or separated by a comma. Only rows where the condition evaluates to TRUE are kept. Rows of monitor$meta where the condition evaluates to NA are dropped. Associated olumns of monitor$data are also dropped for internal consistency in the returned *mts_monitor* object.

monitor_filter() is an alias for monitor_filterMeta().

Usage

```
monitor_filterMeta(monitor, ...)

monitor_filter(monitor, ...)
```

Arguments

monitor	*mts_monitor* object.
...	Logical predicates defined in terms of the variables in monitor$meta.

Value

A subset of the incoming mts_monitor. (A list with meta and data dataframes.)

Note

Filtering is done on variables in monitor$meta.

See Also

monitor_filterDate

monitor_filterDatetime

Examples

```
library(AirMonitor)

# Filter based on countyName field
Camp_Fire %>%
  monitor_filter(countyName == "Alameda") %>%
  monitor_timeseriesPlot(main = "All Alameda County Monitors")

# Filter combining two fields
Camp_Fire %>%
  monitor_filter(latitude > 39.5, longitude > -121.5) %>%
```

```
  monitor_getMeta() %>%
  dplyr::pull(locationName)

# Filter using string matching
Camp_Fire %>%
  monitor_filter(stringr::str_detect(locationName, "^San")) %>%
  monitor_getMeta() %>%
  dplyr::pull(locationName)
```

`monitor_fromPWFSLSmoke` *Convert a ws_monitor object from the PWFSLSmoke package*

Description

A **PWFSLSmoke** package *ws_monitor* object is enhanced and modified so that it becomes a valid *mts_monitor* object. This is a lossless operation and can be reversed with `monitor_toPWFSLSmoke()`.

Usage

```
monitor_fromPWFSLSmoke(ws_monitor = NULL)
```

Arguments

`ws_monitor`	*ws_monitor* object. (A list with `meta` and `data` dataframes.)

Value

A *mts_monitor* object.

`monitor_getCurrentStatus` *Get current status of monitors*

Description

This function augments `monitor$meta` with summary information derived from `monitor$data` reflecting recent measurements.

Usage

```
monitor_getCurrentStatus(
  monitor,
  enddate = NULL,
  minHours = 18,
  dayBoundary = c("clock", "LST")
)
```

Arguments

monitor	*mts_monitor* object.
enddate	Time relative to which current status is calculated. By default, it is the latest time in monitor$data$datetime. This time can be given as a POSIXct time, or a string/numeric value in ymd format (*e.g.* 20190301). This time converted to UTC.
minHours	Minimum number of valid hourly records required to calculate yesterday_PM2.5_avg. Days with fewer valid records will be assigned NA.
dayBoundary	Treatment of daylight savings time: "clock" uses daylight savings time as defined in the local timezone, "LST" uses "local standard time" all year round. (See monitor_dailyStatistic() for more details.)

Value

The monitor$meta table augmented with current status information for each time series.

"Last" and "Previous"

The goal of this function is to provide useful information about what happened recently with each time series in the provided *mts_monitor* object. Devices don't always consistently report data, however, and it is not alwlays useful to have NA's reported when there is recent valid data at earlier times. To address this, monitor_getCurrentStatus() uses *last* and *previous* valid times. These are the time when a monitor most recently reported data, and the most recent time of valid data before that, respectively. By reporting on these times, this function ensures that valid data is returned and provides information on how outdated this information is. This information can be used in maps to show AQI colored dots when data is only a few hours old but gray dots when data is older than some threshold.

Calculating latency

According to https://docs.airnowapi.org/docs/HourlyDataFactSheet.pdf a datum assigned to 2pm represents the average of data between 2pm and 3pm. So, if we check at 3:15pm and see that we have a value for 2pm but not 3pm then the data are completely up-to-date with zero latency.

monitor_getCurrentStatus() defines latency as the difference between a time index and the next most recent time index associated with a valid value. If there is no more recent time index, then the difference is measured to the given enddate parameter. Because *mts_monitor* objects are defined on an hourly axis, these differences have units of hours.

For example, if the recorded values for a monitor are [16.2, 15.8, 16.4, NA, 14.0, 12.5, NA, NA, 13.3, NA], then the *last* valid value is 13.3 with an index is 9, and the *previous* valid value is 12.4 with an index of 6. The *last* latency is then 1 (hour before the end), and the *previous* latency is 3 (hours before the last valid value).

Summary data

The table created by monitor_getCurrentStatus() includes per-time series summary information calculated from monitor$data. The additional data fields added to monitor$meta are listed below:

currentStatus_processingTime Time at which this function was run

currentStatus_enddate Time relative to which "currency" is calculated

last_validIndex Row index of the last valid mesurement in `monitor$data`

previous_validIndex Row index of the previous valid measurement in `monitor$data`

last_validTime UTC time associated with `last_validIndex`

previous_validTime UTC time associated with `previous_validIndex`

last_latency Hours between `last_validTime` and `endtime`

previous_latency Hours between `previous_validTime` and`last_validTime`

last_validLocalTimestamp Local time representation of `last_validTime`

previous_validLocalTimestamp Local time representation of `previous_validTime`

last_PM2.5 Last valid PM2.5 measurement

previous_PM2.5 Previous valid PM2.5 measurement

last_nowcast Last valid PM2.5 NowCast value

previous_nowcast Previous valid PM2.5 NowCast value

yesterday_PM2.5_avg Daily average PM2.5 for the day prior to `enddate`

Examples

```
# Fail gracefully if any resources are not available
try({

library(AirMonitor)

monitor <- airnow_loadLatest()
# TODO:  Needed before rebuilding of v2 database with fullAQSID
monitor$meta$fullAQSID <- paste0("840", monitor$meta$AQSID)

currentStatus <-
  monitor %>%
  monitor_filter(stateCode == "WA") %>%
  monitor_getCurrentStatus()

}, silent = FALSE)
```

`monitor_getDataFrame` *Extract dataframes from* mts_monitor *objects*

Description

These functions are convenient wrappers for extracting the dataframes that comprise a *mts_monitor* object. These functions are designed to be useful when manipulating data in a pipeline using `%>%`.

Below is a table showing equivalent operations for each function.

Function	Equivalent Operation
monitor_getData(monitor)	monitor$data
monitor_getMeta(monitor)	monitor$meta

Usage

```
monitor_getData(monitor)

monitor_getMeta(monitor)
```

Arguments

monitor	*mts_monitor* object to extract dataframe from.

Value

A dataframe from the given *mts_monitor* object.

---

monitor_getDistance *Calculate distances from* mts_monitor *locations to a location of interest*

Description

This function returns the distances (meters) between monitor locations and a location of interest. These distances can be used to create a mask identifying monitors within a certain radius of the location of interest.

Usage

```
monitor_getDistance(
  monitor = NULL,
  longitude = NULL,
  latitude = NULL,
  measure = c("geodesic", "haversine", "vincenty", "cheap")
)
```

Arguments

monitor	*mts_monitor* object.
longitude	Longitude of the location of interest.
latitude	Latitude of the location of interest.
measure	One of "geodesic", "haversine" "vincenty", or "cheap".

Value

Named vector of distances (meters) with each distance identified by deviceDeploymentID.

Note

The measure "cheap" may be used to speed things up depending on the spatial scale being considered. Distances calculated with measure = "cheap" will vary by a few meters compared with those calculated using measure = "geodesic".

Examples

```
library(AirMonitor)

# Walla Walla
longitude <- -118.3302
latitude <- 46.065

distance <- monitor_getDistance(NW_Megafires, longitude, latitude)
closestIndex <- which(distance == min(distance))

# Distance in meters
distance[closestIndex]

# Monitor core metadata
str(NW_Megafires$meta[closestIndex, AirMonitor::coreMetadataNames])
```

---

monitor_isEmpty *Test for an empty* mts_monitor *object*

Description

This function returns true under the following conditions:

- no time series: ncol(monitor$data) == 1
- no time series records: nrow(monitor$data) == 0
- all timeseries values are NA

This makes for more readable code in functions that need to test for this.

Usage

```
monitor_isEmpty(monitor)
```

Arguments

monitor	*mts_monitor* object

Value

Invisibly returns TRUE if no data exist in mts_monitor, FALSE otherwise.

monitor_isValid *Test* mts_monitor *object for correct structure*

Description

The mts_monitor is checked for the presence of core meta and data columns.

Core meta columns include: (TODO: complete this list)

- deviceDeploymentID – unique identifier (see **MazmaLocationUtils**)
- deviceID – device identifier
- locationID – location identifier (see **MazmaLocationUtils**)
- locationName – English language name
- longitude – decimal degrees E
- latitude – decimal degrees N
- elevation – elevation of station in m
- countryCode – ISO 3166-1 alpha-2
- stateCode – ISO 3166-2 alpha-2
- timezone – Olson time zone

Core data columns include:

- datetime – measurement time (UTC)

Usage

```
monitor_isValid(monitor = NULL, verbose = FALSE)
```

Arguments

monitor	*mts_monitor* object
verbose	Logical specifying whether to produce detailed warning messages.

Value

Invisibly returns TRUE if mts_monitor has the correct structure, FALSE otherwise.

`monitor_leaflet` *Leaflet interactive map of monitor locations*

Description

This function creates interactive maps that will be displayed in RStudio's 'Viewer' tab. The `slice` argument is used to collapse a *mts_monitor* timeseries into a single value. If `slice` is an integer, that row index will be selected from the `monitor$data` dataframe. If `slice` is a function (unquoted), that function will be applied to the timeseries with the argument `na.rm=TRUE` (e.g. `max(..., na.rm=TRUE)`).

If `slice` is a user defined function, it will be used with argument `na.rm=TRUE` to collapse the time dimension. Thus, user defined functions must accept `na.rm` as an argument.

Usage

```
monitor_leaflet(
  monitor,
  slice = "max",
  radius = 10,
  opacity = 0.7,
  maptype = "terrain",
  extraVars = NULL,
  jitter = 5e-04,
  ...
)
```

Arguments

`monitor`	*mts_monitor* object.
`slice`	Either a formatted time string, a time index, the name of a (potentially user defined) function used to collapse the time axis.
`radius`	radius of monitor circles
`opacity`	opacity of monitor circles
`maptype`	optional name of leaflet ProviderTiles to use, e.g. "terrain"
`extraVars`	Character vector of additional column names from `monitor$meta` to be shown in leaflet popups.
`jitter`	Amount to use to slightly adjust locations so that multiple monitors at the same location can be seen. Use zero or `NA` to see precise locations.
`...`	Additional arguments passed to `leaflet::addCircleMarker()`.

Details

The `maptype` argument is mapped onto leaflet "ProviderTile" names. Current map types include:

1. "roadmap" – "OpenStreetMap"

2. "satellite" – "Esri.WorldImagery"
3. "terrain" – "Esri.WorldTopoMap"
4. "toner" – "Stamen.Toner"

If a character string not listed above is provided, it will be used as the underlying map tile if available. See https://leaflet-extras.github.io/leaflet-providers/ for a list of "provider tiles" to use as the background map.

Value

Invisibly returns a leaflet map of class "leaflet".

Examples

```
## Not run:
library(AirMonitor)
# Fail gracefully if any resources are not available
try({

monitor_loadLatest() %>%
  monitor_filter(stateCode %in% CONUS) %>%
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

---

monitor_load *Load monitoring data from all sources*

Description

Loads monitoring data for a given time range. Data from AirNow, AIRSIS and WRCC are combined into a single *mts_monitor* object.

Archival datasets are combined with 'daily' and 'latest' datasets as needed to satisfy the requested date range.

Usage

```
monitor_load(
  startdate = NULL,
  enddate = NULL,
  timezone = NULL,
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  epaPreference = c("airnow", "epa_aqs")
)
```

Arguments

startdate	Desired start datetime (ISO 8601).
enddate	Desired end datetime (ISO 8601).
timezone	Olson timezone used to interpret dates.
archiveBaseUrl	Base URL for monitoring v2 data files.
archiveBaseDir	Local base directory for monitoring v2 data files.
QC_negativeValues	Type of QC to apply to negative values. files are available from both 'epa' and 'airnow'.
epaPreference	Preferred data source for EPA data when annual data files are available from both 'epa_aqs' and 'airnow'.

Value

A *mts_monitor* object with PM2.5 monitoring data. (A list with `meta` and `data` dataframes.)

See Also

monitor_loadAnnual

monitor_loadDaily

monitor_loadLatest

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

wa <-
  monitor_load(20210601, 20211001) %>%
  monitor_filter(stateCode == "WA")

monitor_timeseriesPlot(wa)

}, silent = FALSE)

## End(Not run)
```

`monitor_loadAnnual` *Load annual monitoring data from all sources*

Description

Combine annual data from AirNow, AIRSIS and WRCC:

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

Current year files loaded by this function are updated once per week.

For the most recent data in the last 10 days, use `monitor_loadLatest()`.

For daily updates covering the most recent 45 days, use `monitor_loadDaily()`.

For data extended more than 45 days into the past, use `monitor_load()`.

Usage

```
monitor_loadAnnual(
  year = NULL,
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  epaPreference = c("airnow", "epa_aqs")
)
```

Arguments

`year`	Year [YYYY].
`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`epaPreference`	Preferred data source for EPA data when annual data files are available from both ‘epa_aqs‘ and ‘airnow‘.

Value

A *mts_monitor* object with PM2.5 monitoring data. (A list with `meta` and `data` dataframes.)

Note

This function guarantees that only a single time series will be associated with each `locationID` using the following logic:

1. AirNow data takes precedence over data from AIRSIS or WRCC
2. more recent data takes precedence over older data

This relevant mostly for "temporary" monitors which may be replaced after they are initially deployed. If you want access to all device deployments associated with a specific `locationID`, you can use the provider specific functions: `airnow_loadAnnual`, `airsis_loadAnnual` and `wrcc_loadAnnual`

See Also

`monitor_loadDaily`

`monitor_loadLatest`

Examples

```
## Not run:
library(AirMonitor)
# Fail gracefully if any resources are not available
try({

monitor_loadAnnual() %>%
  monitor_filter(stateCode %in% CONUS) %>%
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

---

monitor_loadDaily *Load daily monitoring data from all sources*

Description

Combine daily data from AirNow, AIRSIS and WRCC:

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated once per day and contain data for the previous 45 days.

For the most recent data in the last 10 days, use `monitor_loadLatest()`.

For data extended more than 45 days into the past, use `monitor_load()`.

Usage

```
monitor_loadDaily(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore")
)
```

Arguments

archiveBaseUrl	Base URL for monitoring v2 data files.
archiveBaseDir	Local base directory for monitoring v2 data files.
QC_negativeValues	Type of QC to apply to negative values.

Value

A *mts_monitor* object with PM2.5 monitoring data. (A list with meta and data dataframes.)

Note

This function guarantees that only a single time series will be associated with each locationID using the following logic:

1. AirNow data takes precedence over data from AIRSIS or WRCC
2. more recent data takes precedence over older data

This relevant mostly for "temporary" monitors which may be replaced after they are initially deployed. If you want access to all device deployments associated with a specific locationID, you can use the provider specific functions: airnow_loadDaily, airsis_loadDaily and wrcc_loadDaily

See Also

monitor_loadAnnual

monitor_loadLatest

Examples

```
## Not run:
library(AirMonitor)
# Fail gracefully if any resources are not available
try({

monitor_loadDaily() %>%
  monitor_filter(stateCode %in% CONUS) %>%
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`monitor_loadLatest` *Load most recent monitoring data from all sources*

Description

Combine recent data from AirNow, AIRSIS and WRCC:

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated multiple times an hour and contain data for the previous 10 days.

For daily updates covering the most recent 45 days, use `monitor_loadDaily()`.

For data extended more than 45 days into the past, use `monitor_load()`.

Usage

```
monitor_loadLatest(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore")
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.

Value

A *mts_monitor* object with PM2.5 monitoring data. (A list with `meta` and `data` dataframes.)

Note

This function guarantees that only a single time series will be associated with each `locationID` using the following logic:

1. AirNow data takes precedence over data from AIRSIS or WRCC
2. more recent data takes precedence over older data

This relevant mostly for "temporary" monitors which may be replaced after they are initially deployed. If you want access to all device deployments associated with a specific `locationID`, you can use the provider specific functions: `airnow_loadLatest`, `airsis_loadLatest` and `wrcc_loadLatest`

See Also

monitor_loadAnnual

monitor_loadDaily

Examples

```
## Not run:
library(AirMonitor)
# Fail gracefully if any resources are not available
try({

monitor_loadLatest() %>%
  monitor_filter(stateCode %in% CONUS) %>%
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`monitor_mutate` *Apply a function to* mts_monitor *time series*

Description

This function works similarly to `dplyr::mutate()` and applies `FUN` to each time series found in `monitor$data`. `FUN` must be a function that accepts a numeric vector as its first argument and returns a vector of the same length.

Usage

```
monitor_mutate(monitor = NULL, FUN = NULL, ...)
```

Arguments

`monitor`	*mts_monitor* object.
`FUN`	Function used to modify time series.
`...`	Additional arguments to be passed to `FUN`.

Value

A modified `mts_monitor` object. (A list with `meta` and `data` dataframes.)

Examples

```
library(AirMonitor)

Carmel_Valley %>%
  monitor_filterDatetime(2016080207, 2016080212) %>%
  monitor_toCSV(includeMeta = FALSE) %>%
  cat()

Carmel_Valley %>%
  monitor_filterDatetime(2016080207, 2016080212) %>%
  monitor_mutate(function(x) { return(x / 2) }) %>%
  monitor_toCSV(includeMeta = FALSE) %>%
  cat()
```

---

monitor_nowcast *Apply NowCast algorithm to* mts_monitor *data*

Description

A NowCast algorithm is applied to the data in in the `monitor` object. The `version` argument specifies the minimum weight factor and number of hours to be used in the calculation.

Available versions include:

1. `pm`: hours = 12, weight = 0.5
2. `pmAsian`: hours = 3, weight = 0.1
3. `ozone`: hours = 8, weight = NA

The default, `version = "pm"`, is appropriate for typical usage.

Usage

```
monitor_nowcast(
  monitor,
  version = c("pm", "pmAsian", "ozone"),
  includeShortTerm = FALSE
)
```

Arguments

`monitor`	*mts_monitor* object.
`version`	Name of the type of nowcast algorithm to be used.
`includeShortTerm`	Logical specifying whether to alcluate preliminary NowCast values starting with the 2nd hour.

Details

This function calculates each hour's NowCast value based on the value for the given hour and the previous N-1 hours, where N is the number of hours appropriate for the `version` argument. For example, if `version = "pm"`, the NowCast value for Hour 12 is based on the data from hours 1-12.

The function returns values when at least two of the previous three hours have data. NA's are returned for hours where this condition is not met.

By default, the funtion will not return a valid value until the Nth hour. If `includeShortTerm = TRUE`, the function will return a valid value after only the 2nd hour (provided, of course, that both hours are valid).

Calculated Nowcast values are truncated to the nearest .1 ug/m3 for 'pm' and nearest .001 ppm for 'ozone' regardless of the precision of the data in the incoming *mts_monitor* object.

Value

A modified `mts_monitor` object. (A list with `meta` and `data` dataframes.)

References

https://en.wikipedia.org/wiki/Nowcast_(Air_Quality_Index)

AQI Technical Assistance Document

`monitor_replaceValues` *Replace* mts_monitor *data with another value*

Description

Use an R expression to identify values for replacement.

The **R** expression given in `filter` is used to identify elements in `monitor$data` that should be replaced. The `datetime` column will be retained unmodified. Typical usage would include

1. replacing negative values with 0
2. replacing unreasonably high values with `NA`

Expressions should use `data` for the left hand side of the comparison.

Usage

```
monitor_replaceValues(monitor = NULL, filter = NULL, value = NULL)
```

Arguments

`monitor`	*mts_monitor* object.
`filter`	RR expression used to identify values for replacement.
`value`	Numeric replacement value.

Value

A modified mts_monitor object. (A list with meta and data dataframes.)

Examples

```
library(AirMonitor)

wa <- monitor_filterMeta(NW_Megafires, stateCode == 'WA')
any(wa$data < 5, na.rm = TRUE)

wa_zero <- monitor_replaceValues(wa, data < 5, 5)
any(wa_zero$data < 5, na.rm = TRUE)
```

---

monitor_select *Subset and reorder time series within an* mts_monitor *object*

Description

This function acts similarly to dplyr::select() working on monitor$data. The returned *mts_monitor* object will contain only those time series identified by id in the order specified.

This can be helpful when using faceted plot functions based on **ggplot** such as those found in the **AirMonitorPlots** package.

Usage

```
monitor_select(monitor, id)

monitor_reorder(monitor, id)
```

Arguments

monitor	*mts_monitor* object.
id	Vector of deviceDeploymentIDs.

Value

A reordered (subset) of the incoming *mts_monitor* object. (A list with meta and data dataframes.)

See Also

monitor_filterMeta

`monitor_selectWhere` *Data-based subsetting of time series within an* mts_monitor *object.*

Description

Subsetting of `monitor` acts similarly to `tidyselect::where()` working on `monitor$data`. The returned *mts_monitor* object will contain only those time series where `FUN` applied to the time series data returns `TRUE`.

Usage

```
monitor_selectWhere(monitor, FUN)
```

Arguments

`monitor`	*mts_monitor* object.
`FUN`	A function applied to time series data that returns TRUE or FALSE.

Value

A subset of the incoming *mts_monitor* object. (A list with `meta` and `data` dataframes.)

See Also

monitor_select

Examples

```
library(AirMonitor)

# Show all Camp_Fire locations
Camp_Fire$meta$locationName

# Use package US_AQI data for HAZARDOUS
name <- US_AQI$names_eng[6]
threshold <- US_AQI$breaks_PM2.5[6]

# Find HAZARDOUS locations
worst_sites <-
  Camp_Fire %>%
  monitor_selectWhere(
    function(x) { any(x >= threshold, na.rm = TRUE) }
  )

# Show the worst locations
worst_sites$meta$locationName
```

`monitor_timeInfo` *Get time related information for a monitor*

Description

Calculate the local time for a monitor, as well as sunrise, sunset and solar noon times, and create several temporal masks.

The returned dataframe will have as many rows as the length of the incoming UTC `time` vector and will contain the following columns:

- `localStdTime_UTC` – UTC representation of local **standard** time
- `daylightSavings` – logical mask = TRUE if daylight savings is in effect
- `localTime` – local clock time
- `sunrise` – time of sunrise on each localTime day
- `sunset` – time of sunset on each localTime day
- `solarnoon` – time of solar noon on each localTime day
- `day` – logical mask = TRUE between sunrise and sunset
- `morning` – logical mask = TRUE between sunrise and solarnoon
- `afternoon` – logical mask = TRUE between solarnoon and sunset
- `night` – logical mask = opposite of day

Usage

```
monitor_timeInfo(monitor = NULL, id = NULL)
```

Arguments

`monitor`	*mts_monitor* object.
`id`	`deviceDeploymentID` used to select a single time series found in `monitor`. – optional if `monitor` only has one time series.

Details

While the **lubridate** package makes it easy to work in local timezones, there is no easy way in R to work in "Local Standard Time" (LST) (*i.e. never shifting to daylight savings*) as is often required when working with air quality data. US EPA regulations mandate that daily averages be calculated based on LST.

The `localStdTime_UTC` is primarily for use internally and provides an important tool for creating LST daily averages and LST axis labeling.

Value

A dataframe with times and masks.

Examples

```
library(AirMonitor)

carmel <-
  Carmel_Valley %>%
  monitor_filterDate(20160801, 20160810)

# Create timeInfo object for this monitor
ti <- monitor_timeInfo(carmel)

# Subset the data based on day/night masks
data_day <- carmel$data[ti$day,]
data_night <- carmel$data[ti$night,]

# Build two monitor objects
carmel_day <- list(meta = carmel$meta, data = data_day)
carmel_night <- list(meta = carmel$meta, data = data_night)

# Plot them
carmel_day %>%
  monitor_timeseriesPlot(
    pch = 8,
    col = "goldenrod",
    shadedNight = TRUE
  )

carmel_night %>%
  monitor_timeseriesPlot(
    add = TRUE,
    pch = 16,
    col = "darkblue"
  )
```

`monitor_timeRange` *Get the time range for a monitor*

Description

This function is a wrapper for `range(monitor$data$datetime)` and is convenient for use in data pipelines.

Dates will be returned in the timezone associated with `monitor$data$datetime` which is typically "UTC" unless `timezone` is specified.

Usage

```
monitor_timeRange(monitor = NULL, timezone = NULL)
```

Arguments

monitor	*mts_monitor* object.
timezone	Olson timezone for the returned dates.

Value

A vector containing the minimum and maximum times of a *mts_monitor* object.

Examples

```
Carmel_Valley %>%
  monitor_timeRange(timezone = "America/Los_Angeles")
```

---

monitor_timeseriesPlot

Create timeseries plot

Description

Creates a time series plot of data from a *mts_monitor* object. By default, points are plotted as semi-transparent squares. All data values are plotted from all monitors found in the *mts_monitor* object.

Reasonable defaults are chosen for annotations and plot characteristics. Users can override any defaults by passing in parameters accepted by `graphics::plot.default`.

Usage

```
monitor_timeseriesPlot(
  monitor = NULL,
  id = NULL,
  shadedNight = FALSE,
  add = FALSE,
  addAQI = FALSE,
  palette = c("EPA", "subdued", "deuteranopia"),
  opacity = NULL,
  ...
)
```

Arguments

monitor	*mts_monitor* object.
id	`deviceDeploymentID` used to limit plotting to a single time series found in `monitor`.
shadedNight	Logical specifying whether to add nighttime shading.
add	Logical specifying whether to add to the current plot.

addAQI	Logical specifying whether to add visual AQI decorations.
palette	Named color palette to use when adding AQI decorations.
opacity	Opacity to use for points. By default, an opacity is chosen based on the number of points so that trends are highlighted while outliers diminish in visual importance as the number of points increases.
...	Additional arguments to be passed to `graphics::plot.default()`.

Value

No return value. This function is called to draw an air quality time series plot on the active graphics device.

Examples

```
library(AirMonitor)

# Single monitor
Carmel_Valley %>%
  monitor_timeseriesPlot()

# Multiple monitors
Camp_Fire %>%
  monitor_filter(countyName == "Alameda") %>%
  monitor_timeseriesPlot(main = "All Alameda County Monitors")

# Standard extras
Carmel_Valley %>%
  monitor_timeseriesPlot(
    shadedNight = TRUE,
    addAQI = TRUE
  )
addAQILegend()

# Fancy plot based on pm2.5 values
pm2.5 <- Carmel_Valley$data[,2]
Carmel_Valley %>%
  monitor_timeseriesPlot(
    shadedNight = TRUE,
    pch = 16,
    cex = pmax(pm2.5 / 100, 0.5),
    col = aqiColors(pm2.5),
    opacity = 0.8
  )
addAQILegend(pch = 16, cex = 0.6, bg = "white")
```

---

monitor_toCSV *Convert monitor data as CSV*

Description

Converts the contents of the `monitor` argument to CSV. By default, the output is a text string with "human readable" CSV that includes both `meta` and `data`. When saved as a file, this format is useful for point-and-click spreadsheet users who want to have everything on a single sheet.

To obtain a machine parseable CSV string for just the data, you can use `includeMeta = FALSE`. To obtain machine parseable metadata, use `includeData = FALSE`.

Usage

```
monitor_toCSV(monitor, includeMeta = TRUE, includeData = TRUE)
```

Arguments

`monitor`	*mts_monitor* object.
`includeMeta`	Logical specifying whether to include `monitor$meta`.
`includeData`	Logical specifying whether to include `monitor$data`.

Value

CSV formatted text.

Examples

```
library(AirMonitor)

monitor <-
  Carmel_Valley %>%
  monitor_filterDate(20160802, 20160803)

monitor_toCSV(monitor) %>% cat()
monitor_toCSV(monitor, includeData = FALSE) %>% cat()
monitor_toCSV(monitor, includeMeta = FALSE) %>% cat()
```

---

monitor_toPWFSLSmoke *Convert a mts_monitor object to a ws_monitor object for the PWFSLSmoke package*

Description

A *mts_monitor* object is modified so that it becomes a **PWFSLSmoke** package *ws_monitor* object. While some information will be lost, this operation can be reversed with `monitor_fromPWFSLSmoke()`.

Usage

```
monitor_toPWFSLSmoke(monitor = NULL)
```

Arguments

monitor	*mts_monitor* object

Value

A **PWFSLSmoke** *ws_monitor* object. (A list with meta and data dataframes.)

Note

In order to avoid duplicated monitorID values in the returned *ws_monitor* object, the full deviceDeploymentID will be used as the monitorID.

---

monitor_trimDate *Trim a* mts_monitor *object to full days*

Description

Trims the date range of a *mts_monitor* object to local time date boundaries which are *within* the range of data. This has the effect of removing partial-day data records at the start and end of the timeseries and is useful when calculating full-day statistics.

By default, multi-day periods of all-missing data at the beginning and end of the timeseries are removed before trimming to date boundaries. If trimEmptyDays = FALSE all records are retained except for partial days beyond the first and after the last date boundary.

Day boundaries are calculated using the specified timezone or, if NULL, from monitor$meta$timezone.

Usage

```
monitor_trimDate(monitor = NULL, timezone = NULL, trimEmptyDays = TRUE)
```

Arguments

monitor	*mts_monitor* object.
timezone	Olson timezone used to interpret dates.
trimEmptyDays	Logical specifying whether to remove days with no data at the beginning and end of the time range.

Value

A subset of the given *mts_monitor* object. (A list with meta and data dataframes.)

Examples

```
library(AirMonitor)

# Non-day boundaries
monitor <-
  Camp_Fire %>%
  monitor_filterDatetime(
    "2018111502",
    "2018112206",
    timezone = "America/Los_Angeles"
  )

monitor %>%
  monitor_timeRange(timezone = "America/Los_Angeles")

# Trim to full days only
monitor %>%
  monitor_trimDate() %>%
  monitor_timeRange(timezone = "America/Los_Angeles")
```

`NW_Megafires` *NW_Megafires example dataset*

Description

The `NW_Megafires` dataset provides a quickly loadable version of a *mts_monitor* object for practicing and code examples.

Usage

```
NW_Megafires
```

Format

A *mts_monitor* object with 1080 rows and 143 columns of data.

Details

In the summer of 2015, Washington state had several catastrophic wildfires that led to many days of heavy smoke in eastern Washington, Oregon and northern Idaho. The NW_Megafires dataset contains monitoring data for the Pacific Northwest from July 24 through September 06, 2015.

This dataset was generated on 2022-10-28 by running:

```
library(AirMonitor)

NW_Megafires <-
  monitor_loadAnnual(2015, epaPreference = "epa_aqs")
```

```
  monitor_filterMeta(stateCode
  monitor_filterDate(20150724, 20150907, timezone = "America/Los_Angeles")
  monitor_dropEmpty()

save(NW_Megafires, file = "data/NW_Megafires.rda")
```

`pollutantNames` *Names of standard pollutants*

Description

Character string identifiers of recognized pollutant names.

Usage

```
pollutantNames
```

Format

A vector of character strings

Details

pollutantNames

Examples

```
print(coreMetadataNames, width = 80)
```

`QC_invalidateConsecutiveSuspectValues`
Invalidate consecutive suspect values.

Description

Invalidates values within a timeseries that appear "sticky". Some temporary monitoring data has stretches of consecutive values, sometimes well outside the range of reasonable. This QC function identifies these "sticky" stretches and returns the original timeseries data with "sticky" stretches replaced with `NA`.

Usage

```
QC_invalidateConsecutiveSuspectValues(
  x = NULL,
  suspectValues = c(0:10 * 1000, NA),
  consecutiveCount = 2
)
```

Arguments

x	Timeseries data.
suspectValues	Vector of numeric values considered suspect.
consecutiveCount	How many suspectValues must appear in a row before they are invalidated.

Value

Returns x with some values potentially replaced with NA.

US_52 *US state codes*

Description

State codes for the 50 states +DC +PR (Puerto Rico).

```
US_52 <- c( "AK","AL","AZ","AR","CA","CO","CT","DE","FL","GA", "HI","ID","IL","IN","IA","KS","KY","LA
"MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ", "NM","NY","NC","ND","OH","OK","OR","PA","RI","SC"
"SD","TN","TX","UT","VT","VA","WA","WV","WI","WY", "DC","PR" )
```

Usage

```
US_52
```

Format

A vector with 52 elements

Details

US state codes

US_AQI *US EPA AQI Index levels, names, colors and action text*

Description

Official, US EPA AQI levels, names, colors and action text are provided in a list for easy coloring and labeling.

Usage

```
US_AQI
```

Format

A list with named elements

Details

AQI breaks and associated names and colors

Breaks

Breakpoints are given in units reported for each parameter and include:

- `breaks_AQI`
- `breaks_CO`
- `breaks_NO2`
- `breaks_OZONE_1hr`
- `breaks_OZONE_8hr`
- `breaks_PM2.5`
- `breaks_PM10`

Colors

Several different color palettes are provided:

- `colors_EPA` – official EPA AQI colors
- `colors_subdued` – subdued colors fo use with leaflet maps
- `colors_deuteranopia` – color vision impaired colors

Names

Names of AQI categories are provided in several languages identified by the ISO 639-2 alpha-3 code:

- `names_eng`
- `names_spa`

Actions

Text for "actions to protect yourself" are provided for each category in several languages identified by the ISO 639-2 alpha-3 code:

- `actions_eng`
- `actions_spa`

Currently supported languages include English (eng) and Spanish (spa).

AQI breaks are defined at https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-doc pdf and are given in units appropriate for each pollutant.

AQI colors are defined at https://docs.airnowapi.org/aq101

Note

The low end of each break category is used as the breakpoint.

Examples

```
print(US_AQI$breaks_AQI)
print(US_AQI$colors_EPA)
print(US_AQI$names_eng)
print(US_AQI$names_spa)
```

---

wrcc_loadAnnual *Load annual WRCC monitoring data*

Description

Loads pre-generated .rda files containing annual WRCC data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

Current year files loaded by this function are updated once per week.

For the most recent data in the last 10 days, use `wrcc_loadLatest()`.

For daily updates covering the most recent 45 days, use `wrcc_loadDaily()`.

Usage

```
wrcc_loadAnnual(
  year = NULL,
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  QC_removeSuspectData = TRUE
)
```

Arguments

`year`	Year [YYYY].
`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`QC_removeSuspectData`	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with WRCC data. (A list with `meta` and `data` dataframes.)

Note

Some older WRCC timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`wrcc_loadDaily`

`wrcc_loadLatest`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

# See https://en.wikipedia.org/wiki/Snake_River_Complex_Fire

# WRCC monitors during the Snake River Complex Fire
wrcc_loadAnnual(2021) \
  monitor_filter(stateCode \
  monitor_filterDate(20210707, 20210820, timezone = "America/Denver") \
  monitor_timeseriesPlot(
    ylim = c(0, 300),
    xpd = NA,
    addAQI = TRUE,
    main = "WRCC monitors during Snake River Complex Fire"
  )

}, silent = FALSE)

## End(Not run)
```

`wrcc_loadDaily` *Load daily WRCC monitoring data*

Description

Loads pre-generated .rda files containing daily WRCC data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated once per day and contain data for the previous 45 days.

For the most recent data in the last 10 days, use `wrcc_loadLatest()`.

For data extended more than 45 days into the past, use `wrcc_loadAnnual()`.

Usage

```
wrcc_loadDaily(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
 archiveBaseDir = NULL,
 QC_negativeValues = c("zero", "na", "ignore"),
 QC_removeSuspectData = TRUE
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`QC_removeSuspectData`	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with WRCC data. (A list with `meta` and `data` dataframes.)

Note

Some older WRCC timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`wrcc_loadAnnual`

`wrcc_loadDaily`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

wrcc_loadDaily() \
```

```
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

`wrcc_loadLatest` *Load most recent WRCC monitoring data*

Description

Loads pre-generated .rda files containing the most recent WRCC data.

If `archiveDataDir` is defined, data will be loaded from this local archive. Otherwise, data will be loaded from the monitoring data repository maintained by the USFS AirFire team.

The files loaded by this function are updated multiple times an hour and contain data for the previous 10 days.

For daily updates covering the most recent 45 days, use `wrcc_loadDaily()`.

For data extended more than 45 days into the past, use `wrcc_loadAnnual()`.

Usage

```
wrcc_loadLatest(
 archiveBaseUrl = paste0("https://airfire-data-exports.s3.us-west-2.amazonaws.com/",
    "monitoring/v2"),
  archiveBaseDir = NULL,
  QC_negativeValues = c("zero", "na", "ignore"),
  QC_removeSuspectData = TRUE
)
```

Arguments

`archiveBaseUrl`	Base URL for monitoring v2 data files.
`archiveBaseDir`	Local base directory for monitoring v2 data files.
`QC_negativeValues`	Type of QC to apply to negative values.
`QC_removeSuspectData`	Removes monitors determined to be misbehaving.

Value

A *mts_monitor* object with WRCC data. (A list with `meta` and `data` dataframes.)

Note

Some older WRCC timeseries contain only values of 0, 1000, 2000, 3000, ... ug/m3. Data from these deployments pass instrument-level QC checks but these timeseries generally do not represent valid data and should be removed. With `QC_removeSuspectData = TRUE` (the default), data is checked and periods reporting only values of 0:10 * 1000 ug/m3 are invalidated.

Only those personally familiar with the individual instrument deployments should work with the "suspect" data.

See Also

`wrcc_loadAnnual`

`wrcc_loadDaily`

Examples

```
## Not run:
library(AirMonitor)

# Fail gracefully if any resources are not available
try({

wrcc_loadLatest() \
  monitor_leaflet()

}, silent = FALSE)

## End(Not run)
```

Index