Package 'IRISSeismic'

July 21, 2026

Type Package

Version 1.10.0

Title Classes and Methods for Seismic Data Analysis

Depends R (>= 3.1.0)

Imports methods, pracma, RCurl, seismicRoll (>= 1.1.0), signal, stringr, XML, stats

Suggests knitr,rmarkdown

VignetteBuilder knitr

Description Provides classes and methods for seismic data analysis. The base classes and methods are inspired by the python code found in the 'ObsPy' python toolbox <https://github.com/obspy/obspy>. Additional classes and methods support data returned by web services provided by the EarthScope Consortium. <https://service.earthscope.org/>.

Collate Class-Trace.R Class-Stream.R Class-IrisClient.R mseedWrappers.R Utils.R spectralUtils.R zzz.R

License GPL (>= 2)

Repository CRAN

SystemRequirements Java JRE 8 or later

NeedsCompilation yes

Author Jonathan Callahan [aut],
Rob Casey [aut],
Gillian Sharer [aut, cre],
Mary Templeton [aut],
Chad Trabant [ctb, cph] (Author of the bundled libmseed library (LGPL-3)),
H. Philip Crotwell [cph] (Author of the bundled TauP, seisFile, and seedCodec libraries (LGPL-3))

Maintainer Gillian Sharer <gillian.sharer@earthscope.org>

Copyright IRISSeismic authors; libmseed (LGPL-3) by Chad Trabant, IRIS Data Management Center https://github.com/iris-edu/libmseed; TauP, seisFile, and seedCodec (LGPL-3) by H. Philip Crotwell

https://github.com/crotwell; reload4j and Gson (Apache-2.0) by
The Apache Software Foundation and Google Inc. respectively;
see inst/NOTICE for details

Date/Publication 2026-07-21 03:40:02 UTC

Contents

IRISSeismic-package . . . 3
assignQuality . . . 12
basicStats . . . 12
butterworth . . . 14
crossSpectrum . . . 16
DDT . . . 18
envelope . . . 20
eventWindow . . . 21
getAvailability . . . 23
getChannel . . . 25
getDataAvailability . . . 28
getDataselect . . . 30
getDistaz . . . 32
getEvalresp . . . 34
getEvent . . . 35
getGaps . . . 37
getNetwork . . . 38
getRotation . . . 41
getSNCL . . . 42
getStation . . . 43
getTimeseries . . . 46
getTraveltime . . . 48
getUnavailability . . . 50
getUpDownTimes . . . 52
hilbert . . . 53
hilbertFFT . . . 55
IrisClient-class . . . 56
IRISSeismic-deprecated . . . 58
McNamaraBins . . . 58
McNamaraPSD . . . 59
mergeTraces . . . 61
mergeUpDownTimes . . . 62
miniseed2Stream . . . 64
multiplyBy . . . 65
noiseMatrix2PdfMatrix . . . 66
noiseModels . . . 67
psdDF2NoiseMatrix . . . 68
psdList . . . 69
psdList2NoiseMatrix . . . 71
psdPlot . . . 73

psdStatistics 74
readMiniseedFile 76
rms 77
rotate2D 79
slice 80
STALTA 81
Stream-class 84
surfaceDistance 87
taupTraveltime 88
Trace-class 90
TraceHeader-class 92
transferFunctionSpectra 93
triggerOnset 95
unHistogram 96

Index **97**

`IRISSeismic-package` *Classes and methods for seismic data analysis*

Description

This package provides S4 classes for downloading and processing seismological data available from Earthscope Consortium (https://www.earthscope.org) (formerly IRIS) or other data centers offering FDSN web services. Core classes `Trace`, `Stream` and `IrisClient` and their associated methods are inspired by the functionality available in the python ObsPy package (https://github.com/obspy/obspy/wiki/).

Introduction

The "IRISSeismic-intro" vignette gives introductory examples on using the package.

In 2023, IRIS (Incorporated Research Institutions for Seismology) and UNAVCO merged to form EarthScope Consortium. IRIS (now EarthScope) webservices are unchanged but can now be accessed at https://service.earthscope.org as well as https://service.iris.edu.

History

version 1.10.0

- deprecates functions getTraveltime, getDistaz
- added new function taupTraveltime as a replacement for getTraveltime
- getEvent will now accept any parameter in the FDSN event web service specification
- getNetwork, getStation, getChannel will now accept any parameter in the FDSN station web service specification, except 'level' and 'format'

version 1.9.0

- deprecates functions getAvailability and getUnavailability

version 1.8.0

- added fields to IrisClient object for dataselect_site, station_site, availability_site
- added new miniseed2Stream boolean parameter removeZeroSampleTraces (default = TRUE)
- added new methods `assignQuality.Stream` and `assignQuality.Trace`

version 1.7.1

- added http status code 429 to web service retry criteria

version 1.7.0

- added slot 'retries' to the IrisClient class, for use by select API calls
- addition of Class-IrisClient slot for fdsnws event service site
- addition of environmental variable 'IrisClient_event_site' for fdsnws event service site
- addition of environmental variable 'IrisClient_passw' for authorization credentials for restricted data, used by 'getDataselect.IrisClient'
- bring libmseed code into conformance with CRAN compiled code requirements

version 1.6.7

- update maintainer contact information
- update IRIS references to EarthScope
- replaced service.iris.edu with service.earthscope.org
- changed http calls to https

version 1.6.6

- for web service calls that support the nodata=<204|404> option, use nodata=204
- corrected the crossSpectrum documentation, Pxy is the cross-periodogram for ts1 and ts2

version 1.6.5

- examples and vignette updated for better error handling when accessing internet resources

version 1.6.4

- updated to (modified version of) libmseed-2.19.8

version 1.6.3

- Stream object @ timing_quality now averages the values of the miniSEED blockette 1001 timing quality values, instead of summing the blockette 1001 values and dividing by the number of records

version 1.6.2

- getDataselect, modified default time out values
- fixed url in documentation

version 1.6.0

- irisNetrc definition moved inside getDataselect function
- restored getTimeseries function
- default values for class Trace InstrumentSensitivity and SensitivityFrequency changed to NA from 1.0
- added transferFunctionSpectra function

version 1.5.2

- updated getDataAvailablity to use new fdsnws availability web service specification http://service.iris.edu/fdsnws/availa
- fixed bug in getDataAvailability when mergequality=FALSE
- fixed bug in getDataAvailability affecting start/end times, introduced in version 1.5.1
- fixed bug in mergeTraces when trace has gap at end, introduced in version 1.5.1

version 1.5.1

- changed a subset of time format OS to OS6
- fixed bug in mergeTraces when fdsnws/dataselect implementation cuts on records instead of sample

version 1.5.0

- added spacing as an option to getEvalresp
- modified getEvalresp to use IrisClient service_type
- new getDataAvailability() to return dataframe of miniseed data extents in the IRIS archive using IRIS web service http://service.iris.edu/irisws/availability/1/
- minor change to src code to pass CRAN checks

version 1.4.9

- additional error handling
- minor updates to the plot.Trace and plot.Stream functions
- updated src/libmseed to version 2.19.6

version 1.4.8

- updated src/libmseed to version 2.19.5
- fix bug related to leap seconds
- functions that call web services now follow redirects
- some error outputs have changed slightly
- rmsVariance function, na.rm=TRUE calculates data length minus NA values
- rmsVariance.Stream now honors na.rm=TRUE
- getGaps() error handling now checks for negative sample rates
- getEvent, getEvalresp now truncates start and end input times to seconds (time format OS0 instead of OS) to fix error when user set options(digits.secs=) > 3

version 1.4.7

- additional error handling for getDistaz
- added input service_type to IrisClient, defaults to fdsnws
- plot.Trace x-axis labels are "MM dd" instead of days of week for traces > 1 day and < 1 week
- getDataselect will retry once if it encounters http code 401
- additional error handling for spectralUtils

version 1.4.6

- bug fix for IRISSeismic::slice

version 1.4.5

- fixed bug in noiseModels for low noise model results at periods > 10000 seconds
- retry if getEvent returns a service unavailable message

version 1.4.4

- modified error messages for getEvalresp() and getDistaz()

version 1.4.3

- changed getEvent default url from http://earthquake.usgs.gov/fdsnws/event/1/ to https://earthquake.usgs.gov/fdsnws/eve

version 1.4.2

- updated libmseed version to 2.19

version 1.4.1

- updated libmseed version to 2.18
- fix for reading miniseed with out of order records

version 1.4.0

- addition of repository argument to getDataselect and getSNCL, to match change in fdsnws-dataselect web service

version 1.3.9

- fixes compile warning generated by clang
- removes followlocation=TRUE from getDataselect RCurl options

version 1.3.8

- `getDataselect` does not add a quality indicator to url by default. IRIS webservices itself defaults to quality="M"
- `getStation` and `getChannel` do not add includerestricted indicator to url by default. IRIS webservices itself defaults to `TRUE`
- better handling of textConnections

version 1.3.7

- users can now supply instrument response information in the form of frequency, amplitude, phase to the functions `psdStatistics`, `psdList2NoiseMatrix`, `psdPlot`, in place of the getEvalresp webservice call. Function argument order for `psdPlot` is changed.
- added `showMedian` option to `psdPlot`

version 1.3.5

- added ignoreEpoch option to getDataselect

version 1.3.4 – webservices and plotting

- `getEvent` forwards http://service.iris.edu/fdsnws/event/1/ calls to http://earthquake.usgs.gov/fdsnws/event/1/
- `getDistaz` changes output dataframe column name ellipsoid..attrs to ellipsoid.name
- `plot.Trace` allows for user supplied ylab and xlab input

version 1.3.3 – documentation

- Updated documentation and corrected outdated links

version 1.3.2 – bug fix

- noiseModels(), minor correction to the New High Noise Model

version 1.3.1 – bug fixes

- psdStatistics() correctly handles NA values when calculating high and low PDF bin limits and returns pct_above and pct_below vectors of correct length

version 1.3.0 – compatibility with IRIS webservices

- getDistaz() returns new variables from output of http://services.iris.edu/irisws/distaz/1/

version 1.2.2 – PDF bug fix

- psdList2NoiseMatrix() adds 1 second to start time in getEvalresp call to work around a quirk in http://services.iris.edu/irisws/evalresp/1/ webservice that will not return a response if the start time is exactly on a metadata epoch boundary.

version 1.2.1 – PDF

- psdPlot() now compatible with changes to psdStatistics() in previous version. Adds ylo, yhi arguments to customize y-axis limits in plot.

version 1.2.0 – PDF

- psdStatistics() changes method of setting PDF bins from fixed values to bins based on the high and low PSD values and shifts bin centers by 0.5 dB. The result now matches output from http://services.iris.edu/mustang/noise-pdf.

verison 1.1.7 – improved error handling

- getDataselect(), getNetwork(), getStation(), getChannel(), getAvailability(), getEvalresp(), getTraveltime() error handling now report unexpected http status codes.

version 1.1.6 – bug fixes

- getGaps() fixes issues with multiple sample rates and setting minimum gap length.
- mergeTraces.Stream() relaxes criteria for acceptable sample rate jitter.

version 1.1.5 – trace rotation

- rotate2D() changes orthogonality test tolerance from 5 degrees to 3 degrees.

version 1.1.4 – trace rotation

- rotate2D() exits if traces are not orthogonal.

version 1.1.3 – bug fix

- psdStatistics() fixes bug in calculation of pct_above and pct_below.

version 1.1.1 – bug fixes

- getGaps() minor bug fix.
- mergeTraces.Stream() minor bug fix.

version 1.0.10 – new data request argument and bug fixes

- Imports seismicRoll (>= 1.1.0).
- getGaps() fixes bugs in calculation of initial and final gap of Trace.
- getDataselect(), getSNCL() adds "inclusiveEnd" argument, a logical that determines whether a data point that falls exactly on the requested endtime is included in the Trace.
- libmseed change, when multiple sample rates exist in miniseed records use the mode of all sample rates instead of using the sample rate in the first record.
- psdList() added rule for octave generation for channel codes that start with "V".

version 1.0.9 – Trace class expansion and bug fixes

- Improved error handling for getAvailability(), getChannel(), getDataselect(), getEvalresp(), miniseed2Stream().
- parseMiniSEED.c, unpackdata.c updated. Fixes protection stack overflow issue.
- getGaps() includes a 0.5/sampling_rate tolerance factor.
- miniseed2Stream() uses endtime from parseMiniSEED instead of calculating from the sample rate.
- Trace class now contains slots for optional metadata "latitude", "longitude", "elevation", "depth", "azimuth", "dip", "SensitivityFrequency".
- rotate2D() uses Trace class "azimuth" slot information to identify channel orientation before rotation instead of assuming lead and lag channel from trace input order.

version 1.0.8 – fixes required by ISPAQ

- Removed 'maps' and 'mapdata' from Suggested: packages.
- Changed URL syntax for FDSN web services to use "format=..." instead of "output=...".
- Fixed bug in getSNCL() so that it works when the "quality" argument is missing.

version 1.0.6 – CRAN updates required

- Removed "mode" argument form Trace.as.vector() signature.

version 1.0.4 – name change to IRISSeismic

- Name change required because 'seismic' was recently taken.
- Using explicit references for 'utils' and 'stats' package functions as this is now required for CRAN.

version 1.0.3 – cleanup for submission to CRAN

- Updated libmseed to version 2.16

version 0.2.8.0 – minor tweaks to 0.2.7

- Updated links to IRIS web services in the documentation.
- McNamaraBins() ignores bin #0 (~= DC)
- McNamaraPSD() conversion to dB occurs **after** binning, not before

version 0.2.7.0 – hilbert transform

- New hilbertFFT() function.
- New hilbert() trace method.

version 0.2.6.0 – cross correlation

- Added surfaceDistance() function.
- Added rotate2D() function.

version 0.2.5.0 – channel orientation

- Jumping to version 0.2.5 to match project milestone names.
- Added getSNCL() convenience wrapper for getDataselect() method.
- Added getDistaz() method of IrisClient.
- Added miniseed2Stream() and readMiniseedFile() functions.
- Added getRotation() method of IrisClient.

version 0.2.3.0 – cross spectrum

- Moved McNamaraPSD() from trace method to spectral utility function.
- Added spectral utility functions:
  - crossSpectrum()
  - McNamaraBins()
- All get~ methods that return dataframes now guarantee a default ordering of rows.

version 0.2.2.0 – PSD and friends

- Add dependency on pracma package.
- Use pracma::detrend() function in DDT.Trace().
- Added "increment" parameter to STALTA.Trace().

- Removed STALTA.Trace() algorithm "classic_LR2".
- Fixed URL generation for getEvalresp() when location="".
- Added NamaraPSD.Trace() method.
- Added PSD/PDF utility functions:
  - noiseMatrix2PdfMatrix()
  - noiseModels()
  - psdDF2NoiseMatrix()
  - psdList()
  - psdList2NoiseMatrix()
  - psdStatistics()
  - psdPlot()

version 0.2.1.1 – Bug fix release

- Removed dependcy on signal, XML packages.

version 0.2.1.0 – FDSN web services

- Conversion to FDSN web services including the following new/rewritten methods: `getNetwork`, `getStation`, `getChannel`, `getAvailability`, `getUnavailability`
- Updated version of `getEvent` to return a dataframe with columns named "latitude" and "longitude" for consistency with all other web services
- Updated documentation and Rscripts to match the API changes in the conversion to FDSN web services.
- Removal of all `StationXML` classes in favor of storing that information in slots of the `Trace` class.
- Updates to `Trace` object slots `@Sensor`, `@InstrumentSensitivity` and `@InputUnits` to store information as `character`, `numeric` and `character` instead of `StationXML` classes.
- The `TraceHeader@quality` slot now reflects the data quality returned in the miniSEED record rather than the quality that was requested by getDataselect. (Requests with `quality=B` for "Best" typically return `quality=M`.)
- Improved `STALTA.Trace()` method removes experimental algorithms and now uses C++ code from package `rollSeismic` to calculate rolling means.
- Updated `IrisClient` now uses web services from `http://service.iris.edu` for the following methods: `getDataselect`, `getEvalresp`, `getEvent`

version 0.2.0.0

- Removed PSD methods of `Stream` and `Trace`. PSD algorithms are now part of the PSD metric.
- Improved `mergeTraces.Stream()` method now accepts `fillMethod="fillZero"`.

version 0.1.9.0

- New **rollSeismic** package for fast rolling algorithms implemented in C++/Rcpp.
- New `num_spikes` metric based on `seismicRoll::roll_hampel` outlier detection.

- New `correlation` metric.
- New scripts `glitchMetrics.Rscript`, `correlationMetric.Rscript`, `pressureCorrelation.Rscript`
- New `trace@stats@processing` slot for data processing information.
- New `Stream` methods: `mergeTraces`, `plot`
- Improved `getGaps.Stream()` method properly handles initial and final gaps.
- Improved MCR error messing.

version 0.1.8.0 – IrisClient methods getEvent and getTraveltime, improved SNR metric

version 0.1.7.0 – PSD

version 0.1.6.0 – improved errors, miniSEED parser

version 0.1.5.0 – code cleanup, improved errors, package vignette

version 0.1.4.0 – STA/LTA, upDownTimes, basic plotting

version 0.1.3.0 – SNR, memory profiling

version 0.1.2.0 – ...

version 0.1.1.0 – ...

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

ObsPy: https://github.com/obspy/obspy/wiki/

EarthScope web services: https://service.earthscope.org/

See Also

IrisClient-class, Trace-class, Stream-class,

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2010-02-27 06:45:00", tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:45:00", tz="GMT")

# Get the seismic data
result <- try(st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {

# Extract the first trace, display the metadata and plot it
  tr1 <- st@traces[[1]]
  show(tr1@stats)
  plot(tr1)
}
```

assignQuality *Assign a quality code to a* Stream *or* Trace

Description

The assignQuality methods of Stream and Trace objects return like objects that modify the quality code in the @id slot.

Usage

```
assignQuality(x, quality)
```

Arguments

x	a Trace or Stream object
quality	quality code, one of "R", "D", "Q", "M"

Details

The returned object will have a modified @id slot, where the last character is replaced by quality.

Value

A new Trace or Stream object is returned.

Author(s)

Gillian Sharer <gillian.sharer@earthscope.org>

basicStats *Length, Max, Mean, Median, Min and Standard Deviation*

Description

Basic statistics on the data in Trace and Stream objects.

Usage

```
# length(x)
# max(x, ...)
mean(x, ...)
# median(x, na.rm)
# min(x, ...)
sd(x, na.rm)
parallelLength(x)
parallelMax(x, na.rm)
```

```
parallelMean(x, na.rm)
parallelMedian(x, na.rm)
parallelMin(x, na.rm)
parallelSd(x, na.rm)
```

Arguments

x	a Trace or Stream object
na.rm	a logical specifying whether missing values should be removed
...	arguments to be passed to underlying methods, e.g. the mean function: • na.rm – as above (default=FALSE)

Details

Trace methods

When x is a Trace object, methods length, max, mean, median, min and sd operate on the data slot of the Trace and are equivalent to, e.g., max(x@data, na.rm=FALSE).

Stream methods

When x is a Stream object, methods length, max, mean, median, min and sd are applied to the concatenation of data from every Trace in the Stream, treating this as a single data series.

The parallel~ versions of these methods are available only on Stream objects and return a vector of values, one for each Trace.

By default, the Stream-method versions of these methods use na.rm=FALSE as there should be no missing datapoints in each Trace. The Trace methods default to na.rm=TRUE to accommodate merged traces where gaps have been filled with NAs.

Value

For the simple statistics, a single numeric value is returned or NA if the Trace or Stream has no data.

For the parallel~ versions of these methods, available on Stream objects, a numeric vector is returned of the same length as Stream@traces.

Note

See the R documentation on the respective base functions for further details.

The length.Stream method only counts the number of actual data values in the individual Traces in the Stream object. Missing values associated with the gaps between Traces are not counted.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Get the first trace and generate some statistics
tr1 <- st@traces[[1]]
length(tr1)
max(tr1)
mean(tr1)
sd(tr1)

## End(Not run)
```

`butterworth` *Apply Butterworth filter*

Description

The `butterworth` method of `Trace` objects returns a new `Trace` where data in the `@data` slot have been modified by applying a Butterworth filter.

Usage

```
butterworth(x, n, low, high, type)
```

Arguments

`x`	a `Trace` object
`n`	filter order
`low`	frequency used in low- or stop/band-pass filters
`high`	frequency used in high or stop/band-pass filters
`type`	type of filter – `'low'`, `'high'`, `'pass'` or `'stop'`

Details

This method creates a Butterworth filter with the specified characteristics and applies it to the Trace data.

When only `n` and `low` are specified, a high pass filter is applied. When only `n` and `high` are specified, a low pass filter is applied. When `n` and both `low` and `high` are specified, a band pass filter is applied. To apply a band stop filter you must specify `n`, `low`, `high` and `type='stop'`

Value

A new Trace object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

signal::butter, signal::filter

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Compare to the results in figure 2a of
#
# "Determination of New Zealand Ocean Bottom Seismometer Orientation
#  via Rayleigh-Wave Polarization", Stachnik et al.
#
# http://srl.geoscienceworld.org/content/83/4/704
#
# (note: since publication, ZU.NZ19..BH1 has been renamed BH2 and ZU.NZ19..BH2 has been renamed BH1)

starttime <- as.POSIXct("2009-02-18 22:01:07",tz="GMT")
endtime <- starttime + 630
verticalLines <- starttime + seq(30,630,100)

# Get data
stZ <- getSNCL(iris,"ZU.NZ19..BHZ",starttime,endtime)
st2 <- getSNCL(iris,"ZU.NZ19..BH2",starttime,endtime)
st1 <- getSNCL(iris,"ZU.NZ19..BH1",starttime,endtime)

# Demean, Detrend, Taper
trZ <- DDT(stZ@traces[[1]],TRUE,TRUE,0.05)
tr2 <- DDT(st2@traces[[1]],TRUE,TRUE,0.05)
tr1 <- DDT(st1@traces[[1]],TRUE,TRUE,0.05)

# Bandpass filter
trZ_f <- butterworth(trZ,2,0.02,0.04,type='pass')
tr2_f <- butterworth(tr2,2,0.02,0.04,type='pass')
tr1_f <- butterworth(tr1,2,0.02,0.04,type='pass')

# 3 rows
layout(matrix(seq(3)))

# Plot
plot(trZ_f)
abline(v=verticalLines,col='gray50',lty=2)
plot(tr2_f)
```

```
abline(v=verticalLines,col='gray50',lty=2)
plot(tr1_f)
abline(v=verticalLines,col='gray50',lty=2)

# Restore default layout
layout(1)

## End(Not run)
```

---

crossSpectrum *Cross-Spectral Analysis*

Description

The crossSpectrum() function is based on R's spec.pgram() function and attempts to provide complete results of cross-spectral FFT analysis in a programmer-friendly fashion.

Usage

```
crossSpectrum(x, spans = NULL, kernel = NULL, taper = 0.1,
                          pad = 0, fast = TRUE,
                          demean = FALSE, detrend = TRUE,
                          na.action = stats::na.fail)
```

Arguments

x	multivariate time series
spans	vector of odd integers giving the widths of modified Daniell smoothers to be used to smooth the periodogram
kernel	alternatively, a kernel smoother of class "tskernel"
taper	specifies the proportion of data to taper. A split cosine bell taper is applied to this proportion of the data at the beginning and end of the series
pad	proportion of data to pad. Zeros are added to the end of the series to increase its length by the proportion pad
fast	logical. if TRUE, pad the series to a highly composite length
demean	logical. If TRUE, subtract the mean of the series
detrend	logical. If TRUE, remove a linear trend from the series. This will also remove the mean
na.action	NA action function

Details

The multivariate timeseries passed in as the first argument should be a union of two separate timeseries with the same sampling rate created in the following manner:

```
ts1 <- ts(data1,frequency=sampling_rate)
ts2 <- ts(data2,frequency=sampling_rate)
x <- ts.union(ts1,ts2)
```

The crossSpectrum() function borrows most of its code from R's spec.pgram() function. It omits any plotting functionality and returns a programmer-friendly dataframe of all cross-spectral components generated during Fourier analysis for use in calculating transfer functions.

The naming of cross-spectral components is borrowed from the Octave version of MATLAB's pwelch() function.

Value

A dataframe with the following columns:

`freq`	spectral frequencies
`spec1`	'two-sided' spectral amplitudes for ts1
`spec2`	'two-sided' spectral amplitudes for ts2
`coh`	magnitude squared coherence between ts1 and ts2
`phase`	cross-spectral phase between ts1 and ts2
`Pxx`	periodogram for ts1
`Pyy`	periodogram for ts2
`Pxy`	cross-periodogram for ts1 and ts2

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Octave pwelch() source code

Normalization of Power Spectral Density estimates

See Also

McNamaraPSD

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient")

# Get seismic data
starttime <- as.POSIXct("2011-05-01", tz="GMT")
endtime <- starttime + 3600

st1 <- getDataselect(iris,"CI","PASC","00","BHZ",starttime,endtime)
st2 <- getDataselect(iris,"CI","PASC","10","BHZ",starttime,endtime)
tr1 <- st1@traces[[1]]
tr2 <- st2@traces[[1]]

# Both traces have a sampling rate of 40 Hz
sampling_rate <- tr1@stats@sampling_rate

ts1 <- ts(tr1@data,frequency=sampling_rate)
ts2 <- ts(tr2@data,frequency=sampling_rate)

# Calculate the cross spectrum
DF <- crossSpectrum(ts.union(ts1,ts2),spans=c(3,5,7,9))

# Calculate the transfer function
transferFunction <- DF$Pxy / DF$Pxx
transferAmp <- Mod(transferFunction)
transferPhase <- pracma::mod(Arg(transferFunction) * 180/pi,360)

# 2 rows
layout(matrix(seq(2)))

# Plot
plot(1/DF$freq,transferAmp,type='l',log='x',
     xlab="Period (sec)",
     main="Transfer Function Amplitude")

plot(1/DF$freq,transferPhase,type='l',log='x',
     xlab="Period (sec)", ylab="degrees",
     main="Transfer Function Phase")

# Restore default layout
layout(1)

## End(Not run)
```

DDT *Apply demean, detrend, cosine taper*

Description

The DDT method of Trace objects returns a new Trace where data in the @data slot have been modified. This is typically required before peforming any kind of spectral analysis on the seismic trace.

Usage

```
DDT(x, demean, detrend, taper)
```

Arguments

x	a Trace object
demean	logical specifying whether to deman (default=TRUE)
detrend	logical specifying whether to detrend (default=TRUE)
taper	proportion of the signal to be tapered at each end (default=0.1)

Details

Use taper=0 for no tapering.

Value

A new Trace object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# P-wave onset for a big quake
starttime <- as.POSIXct("2010-02-27 06:30:00", tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:00:00", tz="GMT")
result <- try(st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  tr <- st@traces[[1]]
  trClean <- DDT(tr,TRUE,TRUE,0.1)
  layout(matrix(seq(2)))
  plot(tr)
  abline(h=0,col='gray60')
  mtext("Raw",side=3,line=-2,adj=0.05,col='red')
  plot(trClean)
  abline(h=0,col='gray60')
  mtext("Demean - Detrend - Cosine Taper",line=-2,side=3,adj=0.05,col='red')
}
```

```
# Restore default layout
layout(1)
```

---

envelope *Envelope of a seismic signal*

Description

The envelope method of Trace objects returns a Trace whose data have been replaced with the envelope of the seismic signal.

Usage

```
envelope(x)
```

Arguments

x	a Trace object

Details

Before calculating the envelope, the seismic trace is 'cleaned up' by removing the mean, the trend and by applying a cosine taper. See DDT for more details.

The seismic envelope is defined as:

$$E(t) = \sqrt{T^2(t) + H^2(t)}$$

where $T(t)$ is the seismic trace and $H(t)$ is the Hilbert transform of $T(t)$.

Value

A Trace whose data have been replaced with the envelope of the seismic signal.

Note

This algorithm is adapted from code in the **seewave** package.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2010-02-27 06:00:00", tz="GMT")
endtime <- as.POSIXct("2010-02-27 09:00:00", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]

# Demean, detrend, cosine taper
tr <- DDT(tr)

# Create envelope version of the trace
trenv <- envelope(tr)

# Plot signal data and envelope data
plot(tr@data, type='l', col='gray80')
points(trenv@data, type='l', col='blue')

## End(Not run)
```

`eventWindow` *Return a portion of a trace surrounding an event.*

Description

The `eventWindow` method of `Trace` uses the picker returned by the `STALTA()` method to center a window around the the event detected by the picker.

Usage

```
eventWindow(x, picker, threshold, windowSecs)
```

Arguments

`x`	a `Trace` object
`picker`	a picker as returned by the `STALTA()` method applied to this `Trace`
`threshold`	the threshold at which the picker is 'triggered'
`windowSecs`	the size of the window in secs

Details

This utility function uses the trace method `triggerOnset()` to determine p-wave onset followed by the `slice()` method to return a new `Trace` object of the desired size centered *near* the event onset.

When no threshold value is supplied, the default value is calculated as:

`threshold=quantile(picker,0.999,na.rm=TRUE)`

Value

A new Trace object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

STALTA, triggerOnset

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2002-04-20", tz="GMT")
endtime <- as.POSIXct("2002-04-21", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"US","OXF","","BHZ",starttime,endtime)

# Seismic signal in third trace
tr <- st@traces[[3]]

# Create a picker
picker <- STALTA(tr,3,30)
threshold <- quantile(picker,0.99999,na.rm=TRUE)

# 3 rows
layout(matrix(seq(3)))

# Plot trace and p-wave closeups
closeup1 <- eventWindow(tr,picker,threshold,3600)
closeup2 <- eventWindow(tr,picker,threshold,600)
plot(tr)
plot(closeup1,subsampling=1)
abline(v=length(closeup1)/2, col='red')
plot(closeup2,subsampling=1)
abline(v=length(closeup2)/2, col='red')

# Restore default layout
layout(1)

## End(Not run)
```

getAvailability *Retrieve Channel metadata from EarthScope*

Description

This method is deprecated. It relies on the EarthScope fdsnws/station web service parameter "matchtimeseries" which is no longer supported. Without "matchtimeseries", this method returns the same dataframe results as `getChannel`.

The `getAvailability` method obtains channel metadata for available channels from the EarthScope `station` web service and returns it in a dataframe.

Usage

```
getAvailability(obj, network, station, location, channel,
                starttime, endtime, includerestricted,
                latitude, longitude, minradius, maxradius)
```

Arguments

`obj`	`IrisClient` object
`network`	character string with the two letter seismic network code
`station`	character string with the station code
`location`	character string with the location code
`channel`	character string with the three letter channel code
`starttime`	POSIXct class specifying the starttime (GMT)
`endtime`	POSIXct class specifying the endtime (GMT)
`includerestricted`	optional logical identifying whether to report on restricted data (default=`FALSE`)
`latitude`	optional latitude used when specifying a location and radius
`longitude`	optional longitude used when specifying a location and radius
`minradius`	optional minimum radius used when specifying a location and radius
`maxradius`	optional maximum radius used when specifying a location and radius

Details

The `getAvailability` method uses the station web service to obtain data for all available channels that meet the criteria defined by the arguments and returns that data in a dataframe. Each row of the dataframe represents a unique channel-epoch. This method is equivalent to the `getChannel` method with the following additional parameters attached to the url:

```
&includeavailability=true&matchtimeseries=true
```

Each of the arguments network, station, location or channel may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise the ascii string that is used for these values is simply inserted into the web service request URL. (For non-available channels use getUnavailability.)

For more details see the web service documentation.

Value

A dataframe with the following columns:

```
network, station, location, channel, latitude, longitude, elevation,
  depth, azimuth, dip, instrument, scale, scalefreq, scaleunits,
  samplerate, starttime, endtime, snclId
```

Rows are ordered by snclId.

The snclId column, eg. "US.OCWA..BHE", is generated as a convenience. It is not part of the normal return from the station web service.

Note: The snclId is not a unique identifier. If the time span of interest crosses an epoch boundary where instrumentation was changed then multiple records (rows) will share the same snclId.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope station web service:

https://service.earthscope.org/fdsnws/station/1/

This implementation was inspired by the functionality in the obspy get_stations() method.

https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_stations.html

See Also

IrisClient-class, getChannel, getUnavailability

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Date of Nisqually quake
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- starttime + 2*24*3600

# Use  getEvent web service to retrieve events in this time period
result <- try(events <- getEvent(iris,starttime,endtime,6.0))
if (inherits(result,"try-error")) {
```

```
  message(geterrmessage())
} else {
  events

# biggest event is Nisqually
  eIndex <- which(events$magnitude == max(events$magnitude))
  e <- events[eIndex[1],]

# Find all BHZ channels collecting data at the time of the quake and within
# 5 degrees of the quake epicenter
  result <- try(channels <- getAvailability(iris,"*","*","*","BHZ",starttime,endtime,
                           lat=e$latitude,long=e$longitude,maxradius=5))
  if (inherits(result,"try-error")) {
    message(geterrmessage())
  } else {
    channels
  }
}
```

---

getChannel *Retrieve Channel metadata from EarthScope or other FDSN Data Center*

Description

The `getChannel` method obtains channel metadata from the EarthScope (or other Data center) `station` web service and returns it in a dataframe.

Usage

```
getChannel(
  obj,
  network,
  station,
  location,
  channel,
  starttime,
  endtime,
  includerestricted,
  latitude,
  longitude,
  minradius,
  maxradius,
  ...)
```

Arguments

`obj`	`IrisClient` object

network	character string with the two letter seismic network code
station	optional character string with the station code
location	optional character string with the location code
channel	optional character string with the three letter channel code
starttime	optional POSIXct class specifying the starttime (GMT)
endtime	optional POSIXct class specifying the endtime (GMT)
includerestricted	optional logical identifying whether to report on restricted data
latitude	optional latitude used when specifying a location and radius
longitude	optional longitude used when specifying a location and radius
minradius	optional minimum radius used when specifying a location and radius
maxradius	optional maximum radius used when specifying a location and radius
...	optional other parameters passed to the station service

Details

The getChannel method uses the station web service to obtain data for all channels that meet the criteria defined by the arguments and returns that data in a dataframe. Each row of the dataframe represents a unique channel-epoch.

Each of the arguments network, station, location or channel may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise the ascii string that is used for these values is simply inserted into the web service request URL.

For more details see the webservice documentation.

Value

A dataframe with the following columns:

```
network, station, location, channel, latitude, longitude, elevation,
  depth, azimuth, dip, instrument, scale, scalefreq, scaleunits,
  samplerate, starttime, endtime, snclId
```

Rows are ordered by snclId.

The snclId column, eg. "US.OCWA..BHE", is generated as a convenience. It is not part of the normal return from the station web service.

Note: The snclIds is not a unique identifier. If the time span of interest crosses an epoch boundary where instrumentation was changed then multiple records (rows) will share the same snclId.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope station webservice:

https://service.earthscope.org/fdsnws/station/1/

This implementation was inspired by the functionality in the ObsPy get_stations() method.

https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_stations.html

See Also

IrisClient-class, getAvailability, getUnavailability

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Date of Nisqually quake
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- starttime + 2*24*3600

# Use the getEvent web service to determine what events happened in this time
# period
result <- try(events <- getEvent(iris,starttime,endtime,6.0))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  events

# biggest event is Nisqually
  eIndex <- which(events$magnitude == max(events$magnitude))
  e <- events[eIndex[1],]
}

# Which stations in the US network are within 5 degrees of the quake epicenter?
result <- try(stations <- getStation(iris,"US","*","*","BHZ",starttime,endtime,
                          lat=e$latitude,long=e$longitude,maxradius=5))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  stations
}

# Get some detailed information on any BHZ channels at the "Octopus Mountain"
# station
result <- try(channels <-
  getChannel(iris, "US", "OCWA", "*", "BHZ", starttime, endtime))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  channels
}
```

getDataAvailability *Retrieve miniseed trace extents from EarthScope*

Description

The getDataAvailability method obtains miniseed time extents from the EarthScope availability web service and returns it in a dataframe.

Usage

```
getDataAvailability(obj, network, station, location, channel,starttime, endtime,
                    mergequality, mergesamplerate, mergeoverlap, mergetolerance,
                     includerestricted, excludetoolarge)
```

Arguments

obj	IrisClient object
network	character string with the two letter seismic network code
station	character string with the station code
location	character string with the location code
channel	character string with the three letter channel code
starttime	POSIXct class specifying the starttime (GMT)
endtime	POSIXct class specifying the endtime (GMT)
mergequality	optional logical identifying if timespans with differing qualities are grouped together (default=TRUE)
mergesamplerate	optional logical identifying if timespans from data with differing sample rates are grouped together (default=FALSE)
mergeoverlap	optional logical identifying if overlapping timespans are merged together (default=TRUE)
mergetolerance	optional numeric. Time spans separated by less than or equal to the mergetolerance value in seconds are merged together. To have an effect, the mergetolerance value must be larger than 1.5 times the sample period. This implements the mergegaps option in the fdsnws availability web service specification.
includerestricted	optional logical identifying whether to report on restricted data (default=FALSE)
excludetoolarge	optional logical, if TRUE sets the fdsnws availability web service option "limit=500000". default=TRUE

Details

The `getDataAvailability` method uses the FDSNWS availability service to obtain start and end-times for all continuous trace segments in the IRIS (or other) archive for the requested network, station, location, channels and returns that data in a dataframe. Each row of the dataframe represents a unique data trace extent.

Each of the arguments `network`, `station`, `location` or `channel` may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise the ascii string that is used for these values is simply inserted into the web service request URL.

For more details see the web service documentation.

Value

A dataframe with the following columns:

mergequality=TRUE and mergesamplerate=FALSE (defaults):

```
network, station, location, channel, samplerate, starttime, endtime, snclId
```

mergequality=TRUE and mergesamplerate=TRUE:

```
network, station, location, channel, starttime, endtime, snclId
```

mergequality=FALSE and mergesamplerate=FALSE:

```
network, station, location, channel, quality, samplerate, starttime, endtime, snclId
```

mergequality=FALSE and mergesamplerate=TRUE:

```
network, station, location, channel, quality, starttime, endtime, snclId
```

Rows are ordered by `snclId`.

The snclId column, eg. "US.OCWA..BHE", is generated as a convenience. It is not part of the normal return from the station web service.

Author(s)

Gillian Sharer <gillian.sharer@earthscope.org>

References

The EarthScope station web service:

https://service.earthscope.org/fdsnws/availability/1/ https://service.earthscope.org/ph5ws/availability/1/

See Also

IrisClient-class

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2018-01-01",tz="GMT")
endtime <- as.POSIXct("2019-01-01",tz="GMT")
network <- "TA"
station <- "M22K"
channel <- "BHZ"

result <- try(traceList <- getDataAvailability(iris,network,station,"*",channel,starttime,endtime))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  traceList
}
```

---

getDataselect *Retrieve seismic data from EarthScope or other FDSN Data Center*

Description

The `getDataselect` method makes a request of the EarthScope (or other Data Center) `dataselect` webservice and returns a `Stream` object in which individual `Traces` have been sorted by start time.

Usage

```
getDataselect(
  obj,
  network,
  station,
  location,
  channel,
  starttime,
  endtime,
  ...
)
```

Arguments

`obj`	`IrisClient` object
`network`	character string with the two letter seismic network code
`station`	character string with the station code

location	character string with the location code
channel	character string with the three letter channel code
starttime	POSIXct class specifying the starttime (GMT)
endtime	POSIXct class specifying the endtime (GMT)
...	optional arguments quality optional character string identifying the quality. EarthScope webservices defaults to quality="M". repository optional character string identifying whether to exclusively search primary archive or realtime collection buffers. Acceptable values are "primary" or "realtime". If not specified, EarthScope webservices defaults to both repositories. inclusiveEnd optional logical determining whether the endtime is inclusive (default = TRUE) ignoreEpoch optional logical defining behavior when multiple epochs are encountered (default = FALSE)

Details

This is the primary method for retrieving seismic data. Data requests are made through the dataselect webservice and returned data are parsed using the internal miniseed2Stream() function.

If the location argument contains an empty string to specify a 'blank' location code, a location code of "--" will be used in the dataselect request URL. (See dataselect documentation.)

If inclusiveEnd=FALSE, then getDataselect will subtract 0.000001 seconds from the endtime before passing the value to the dataselect webservice. An endtime of, e.g., as.POSIXct("2016-01-03", tz="GMT") will be passed into dataselect as end=2016-01-02T23:59:59.999999. A data sample at time 2016-01-03T00:00:00 will not be returned unless inclusiveEnd=TRUE.

Error returns from the webservice will stop evaluation and generate an error message.

Sometimes the station webservice will return multiple records for the same SNCL, each with a different scale or starttime. These represent different *epochs* with potentially different metadata parameters for the SNCL and, by default, will cause a 'Multiple epochs' error message to be generated.

Handling all possible metadata differences so that the data may be merged is beyond the scope of this package. Instead, to avoid errors, users may specify ignoreEpoch=TRUE in which case the very first SNCL-epoch encountered will be used and all others will be discarded.

For access to restricted data, getDataselect will look for a system environmental variable "IrisClient_passw" which should be a string in the form "login:password". Alternatively, getDataselect will look for system environmental variable "IrisClient_netrc" which should point to a .netrc authentication file.

Value

A new Stream object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope dataselect webservice:

https://service.earthscope.org/fdsnws/dataselect/1/

This implementation is similar in functionality to the obspy dataselect function:

https://docs.obspy.org/_modules/obspy/clients/fdsn/client.html

See Also

getSNCL, IrisClient-class

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")
options("nanotimeFormat"="%Y-%m-%dT%H-%M-%E9S")

# Use getDataselect to request data for II.JTS.00.BHZ
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- as.POSIXct("2001-03-01",tz="GMT")

st <- getDataselect(iris, "II","JTS","00","BHZ",starttime, endtime,
                    inclusiveEnd=FALSE,ignoreEpoch=TRUE)

# Display structure of trace(s)

str(st)

# Plot trace
plot(st)

## End(Not run)
```

---

getDistaz *Retrieve great circle distance information from EarthScope*

Description

Deprecated due to the upcoming retirement of the EarthScope `distaz` web service.

The `getDistaz` method obtains great circle distance data from the EarthScope `distaz` web service.

Usage

```
getDistaz(obj, latitude, longitude, staLatitude, staLongitude)
```

Arguments

obj	an IrisClient object
latitude	latitude of seismic event
longitude	longitude of seismic event
staLatitude	latitude of seismic station
staLongitude	longitude of seismic station

Details

The distance-azimuth service will calculate the great-circle angular distance, azimuth, and back azimuth between two geographic coordinate pairs. Azimuth and back azimuth are measured clockwise from North.

Value

A dataframe with the following columns:

```
ellipsoid.semiMajorAxis, ellipsoid.flattening, ellipsoid.name, fromlat, fromlon, tolat, tolon,
  azimuth,backAzimuth, distance, distanceMeters
```

Where fromlat is the event latitude, fromlon is the event longitude, tolat is the station latitude, and tolon is the station longitude. azimuth, backAzimuth, and distance are measured in degrees. distanceMeters is distance in meters. ellipsoid.semiMajorAxis, ellipsoid.flattening, and ellipsoid.name refer to the World Geodetic System standard coordinate system version used to correct for ellipticity when converting to geocentric latitudes.

Only a single row is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope distaz webservice:

https://service.earthscope.org/irisws/distaz/1/

See Also

IrisClient-class

`getEvalresp` *Retrieve instrument response information from EarthScope*

Description

The `getEvalresp` method obtains instrument response data from the EarthScope `evalresp` web-service.

Usage

```
getEvalresp(obj, network, station, location, channel,
            time, minfreq, maxfreq, nfreq, units, output, spacing)
```

Arguments

`obj`	an `IrisClient` object
`network`	character string with the two letter seismic network code
`station`	character string with the station code
`location`	character string with the location code
`channel`	character string with the three letter channel code
`time`	POSIXct class specifying the time at which response is evaluated (GMT)
`minfreq`	optional minimum frequency at which response will be evaluated
`maxfreq`	optional maximum frequency at which response will be evaluated
`nfreq`	optional number of frequencies at which response will be evaluated
`units`	optional code specifying unit conversion
`output`	optional code specifying output type (default="fap")
`spacing`	optional code specifying spacing, accepted values are "lin" or "linear", "log" or "logarithmic" (default="log")

Details

The `evalresp` webservice responds to requests with data that can be used to remove instrument response from a seismic signal.

Each of `network`, `station` or `channel` should contain a valid code without wildcards. The ascii string that is used for these values is simply passed through to `evalresp`.

If the `location` argument contains an empty string to specify a 'blank' location code, a location code of `"--"` will be used in the dataselect request URL. (See dataselect documentation.)

The response from `evalresp` is converted into a dataframe with rows in order of increasing frequency.

Value

For `output="fap"`, a dataframe with columns named:

`freq, amp, phase`

For `output="cs"`, a dataframe with columns named:

`freq, real, imag`

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope evalresp webservice:

https://service.earthscope.org/irisws/evalresp/1/

See Also

IrisClient-class,

---

getEvent *Retrieve seismic event information from the USGS NEIC*

Description

The `getEvent` method obtains seismic event data from the USGS NEIC `event` webservice or other FDSN `event` webservice.

Usage

```
getEvent(obj, starttime, endtime, minmag, maxmag, magtype,
         mindepth, maxdepth, ...)
```

Arguments

`obj`	an `IrisClient` object
`starttime`	optional POSIXct class limiting results to events occurring after starttime (GMT)
`endtime`	optional POSIXct class limiting results to events occurring before endtime (GMT)
`minmag`	optional minimum magnitude
`maxmag`	optional maximum magnitude
`magtype`	optional magnitude type
`mindepth`	optional minimum depth (km)
`maxdepth`	optional maximum depth (km)
`...`	optional other arguments

Details

The `getEvent` method uses the event web service to obtain data for all events that meet the criteria defined by the arguments and returns that data in a dataframe. Each row of the dataframe represents a unique event.

`getEvent` calls to the IRIS event webservice now go to https://earthquake.usgs.gov/fdsnws/event/1/. If `obj@site` is something other than "https://service.earthscope.org", getEvent will point to `obj@site`/fdsnws/event/1/. The event service must be able to output format=text.

The USGS NEIC event service defaults to 'NOW - 30 days' if starttime is not provided. It defaults to 'present time' if endtime is not provided.

The optional other arguments are any other parameters accepted by the event web service being used. A full list of available parameters for the USGS NEIC event service can be found at: https://earthquake.usgs.gov/fdsnws/event/1/

Value

A dataframe with the following columns:

```
eventId ,time, latitude, longitude, depth, author, cCatalog, contributor,
  contributorId, magType, magnitude, magAuthor, eventLocationName
```

Rows are ordered by `time`.

NOTE: column names are identical to the names returned from the event web service with the exception of "latitude" for "lat" and "longitude" for "lon". The longer names are used for internal consistency – all other web services return columns named "latitude" and "longitude".

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The USGS NEIC event web service: https://earthquake.usgs.gov/fdsnws/event/1/

See Also

IrisClient-class,

Examples

```
  ## Not run:
# NOTE:  'maps' and 'mapdata' packages must be installed
#require(maps)
#require(mapdata)

# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Get events > mag 5.0 with a minimum depth of 20km over a week in June of 2012
```

```
starttime <- as.POSIXct("2012-06-21", tz="GMT")
endtime <- starttime + 3600 * 24 * 7
result <- try(events <- getEvent(iris, starttime, endtime, minmag=5.0, mindepth=20))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {

# Look at all events
  print(paste(nrow(events),"earthquakes found with magnitude > 5.0 and depth >= 20 km"))

# Plot events on a map
  #map('world')
  #points(events$longitude, events$latitude, pch=16, cex=1.5, col='red')
  #labels <- paste(" ", as.character(round(events$magnitude,1)), sep="")
  #text(events$longitude, events$latitude, labels=labels, pos=4, cex=1.2, col='red3')
  }

## End(Not run)
```

---

getGaps *Gap analysis*

Description

The getGaps method calculates data dropouts that occur within the requested time range associated with a Stream.

A Stream object returned by getDataselect contains a list of individual Trace objects, each of which is guaranteed to contain a continuous array of data in each Trace@data slot. Each TraceHeader also contains a starttime and an endtime defining a period of uninterrupted data collection.

Data dropouts are determined by examining the requestedStartime and requestedEndtime slots associated with the Stream and the starttime and endtime slots found in the each TraceHeader.

Usage

```
getGaps(x, min_gap)
```

Arguments

x	Stream object
min_gap	minimum gap (sec) below which gaps will be ignored (default=1/sampling_rate)

Details

This method first checks the SNCL id of each Trace to make sure they are identical and generates an error if they are not. Mismatches in the sampling_rate will also generate an error.

The data gaps (in seconds) within a Stream are determined and the associated sampling_rate is used to calculate the number of missing values in each gap. The length of the gaps and nsamples

vectors in the returned list will be one more than the number of Traces (inital gap + gaps between traces + final gap).

Gaps smaller than min_gap are set to 0. Values of min_gap smaller than 1/sampling_rate will be ignored and the default value will be used instead.

Overlaps will appear as gaps with negative values.

Value

A list is returned with the following elements:

- gaps numeric vector of data gaps within a Stream
- nsamples number of missing samples associated with each gap

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Save the gap analysis in a variable
gapInfo <- getGaps(st)

# See what information is availble
names(gapInfo)

# Look at a histogram of data dropouts
hist(gapInfo$nsamples, breaks=50,
     main="Data Gaps in AK.PIN..BHZ Jan 24, 2012",
     xlab="number of missing samples per gap")

## End(Not run)
```

---

getNetwork *Retrieve Network metadata from EarthScope or other FDSN station service*

Description

The getNetwork method obtains network metadata from the EarthScope (or other Data Center) station web service and returns it in a dataframe.

Usage

```
getNetwork(
  obj,
  network,
  station,
  location,
  channel,
  starttime,
  endtime,
  includerestricted,
  latitude,
  longitude,
  minradius,
  maxradius,
  ...)
```

Arguments

obj	IrisClient object
network	optional character string with the two letter seismic network code
station	optional character string with the station code
location	optional character string with the location code
channel	optional character string with the three letter channel code
starttime	optional POSIXct class specifying the starttime (GMT)
endtime	optional POSIXct class specifying the endtime (GMT)
includerestricted	optional logical identifying whether to report on restricted data (default=FALSE)
latitude	optional latitude used when specifying a location and radius
longitude	optional longitude used when specifying a location and radius
minradius	optional minimum radius used when specifying a location and radius
maxradius	optional maximum radius used when specifying a location and radius
...	optional other parameters passed to the station web service

Details

The getNetwork method utilizes the station web service to return data for all stations that meet the criteria defined by the arguments and returns that data in a dataframe. Each row of the dataframe represents a unique network.

Each of the arguments network, station, location or channel may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise, the ascii string that is used for these values is simply inserted into the web service request URL.

For more details see the web service documentation.

Value

A dataframe with the following columns:

`network, description, starttime, endtime, totalstations`

Rows are ordered by `network`.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope station web service:

https://service.earthscope.org/fdsnws/station/1/

This implementation was inspired by the functionality in the ObsPy get_stations() method.

https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_stations.html

See Also

IrisClient-class

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Date of Nisqually quake
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- starttime + 2*24*3600

# Use the getEvent web service to determine what events happened in this time
# period
result <- try(events <- getEvent(iris,starttime,endtime,6.0))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  events

# biggest event is Nisqually
  eIndex <- which(events$magnitude == max(events$magnitude))
  e <- events[eIndex[1],]


# Which seismic networks have BHZ stations within 5 degrees of the quake
# epicenter?
  result <- try(networks <- getNetwork(iris,"*","*","*","BHZ",starttime,endtime,
                           lat=e$latitude,lon=e$longitude,maxradius=5))
  if (inherits(result,"try-error")) {
```

```
    message(geterrmessage())
  } else {
    networks
  }
}
```

---

getRotation *Retrieve rotated seismic data from EarthScope*

Description

The getRotation method makes a request of the EarthScope rotation web service and returns a list of 3 Stream objects.

Usage

```
getRotation(obj, network, station, location, channelSet,
              starttime, endtime, processing)
```

Arguments

obj	IrisClient object
network	character string with the two letter seismic network code
station	character string with the station code
location	character string with the location code
channelSet	the first two characters of the selected source channels
starttime	POSIXct class specifying the starttime (GMT)
endtime	POSIXct class specifying the endtime (GMT)
processing	optional character string with processing commands

Details

The rotation web service returns a triplet of seismic Streams, rotated according to the processing commands.

If the location argument contains an empty string to specify a 'blank' location code, a location code of "--" will be used in the dataselect request URL.

The processing parameter can be used to specify any type of processing supported by the rotation webs service. This string must begin with an ampersand and be ready to be appended to the request url, e.g. processing="&components=ZRT&azimuth=23.1". This gives the user complete control over the number and order of processing commands. (See rotation documentation.)

Error returns from the web service will stop evaluation and generate an error message.

Value

A list of three Stream objects is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope rotation web service:

https://service.earthscope.org/irisws/rotation/1/

See Also

IrisClient-class

---

getSNCL *Retrieve seismic data from EarthScope*

Description

The getSNCL() method is a convenience wrapper for the getSNCL() method and returns a Stream object in which individual Traces have been sorted by start time.

Usage

```
getSNCL(obj, sncl, starttime, endtime, ...)
```

Arguments

obj	IrisClient object
sncl	character string with the SNCL code
starttime	POSIXct class specifying the starttime (GMT)
endtime	POSIXct class specifying the endtime (GMT)
...	optional arguments quality optional character string identifying the quality. IRIS webservices defaults to quality="M". repository optional character string identifying whether to exclusively search primary archive or realtime collection buffers. Acceptable values are "primary" or "realtime". If not specified, IRIS webservices defaults to both repositories. inclusiveEnd optional logical determining whether the endtime is inclusive (default = TRUE) ignoreEpoch optional logical defining behavior when multiple epochs are encountered (default = FALSE)

Details

The SNCL argument should be ordered network-station-location channel, e.g. IU.ANMO.00.LHZ. This argument is split into component parts which are then used in a call to the getSNCL() method.

Value

A new Stream object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope dataselect web service:

https://service.earthscope.org/fdsnws/dataselect/1/

See Also

getDataselect, IrisClient-class

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Use getSNCL to request data for II.JTS.00.BHZ
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- as.POSIXct("2001-03-01",tz="GMT")

st <- getSNCL(iris, "II.JTS.00.BHZ",starttime, endtime, quality="M",
              repository="primary",inclusiveEnd=FALSE,ignoreEpoch=TRUE)

# Display structure of trace(s)

str(st)

# Plot trace
plot(st)

## End(Not run)
```

---

getStation *Retrieve Station metadata from EarthScope or other FDSN Data Center*

Description

The getStation method obtains station metadata from the EarthScope (or other Data Center) station web service and returns it in a dataframe.

Usage

```
getStation(
  obj,
  network,
  station,
  location,
  channel,
  starttime,
  endtime,
  includerestricted,
  latitude,
  longitude,
  minradius,
  maxradius,
  ...
)
```

Arguments

obj	IrisClient object
network	character string with the two letter seismic network code
station	optional character string with the station code
location	optional character string with the location code
channel	optional character string with the three letter channel code
starttime	optional POSIXct class specifying the starttime (GMT)
endtime	optional POSIXct class specifying the endtime (GMT)
includerestricted	optional logical identifying whether to report on restricted data
latitude	optional latitude used when specifying a location and radius
longitude	optional longitude used when specifying a location and radius
minradius	optional minimum radius used when specifying a location and radius
maxradius	optional maximum radius used when specifying a location and radius
...	optional other parameters passed to the station service

Details

The getStation method utilizes the station web service to obtain data for all stations that meet the criteria defined by the arguments and returns that data in a dataframe. Each row of the dataframe represents a unique station.

Each of the arguments network, station, location or channel may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise, the ascii string that is used for these values is simply inserted into the web service request URL.

For more details see the web service documentation.

Value

A dataframe with the following columns:

`network, station, latitude, longitude, elevation, sitename, starttime, endtime`

Rows are ordered by `network-station`.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope station web service:

https://service.earthscope.org/fdsnws/station/1/

This implementation was inspired by the functionality in the ObsPy get_stations() method.

https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_stations.html

See Also

IrisClient-class

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Date of Nisqually quake
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- starttime + 2*24*3600

# Use the getEvent web service to determine what events happened in this time
# period
result <- try(events <- getEvent(iris,starttime,endtime,6.0))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  events

# biggest event is Nisqually
  eIndex <- which(events$magnitude == max(events$magnitude))
  e <- events[eIndex[1],]

# Which stations in the US network are within 5 degrees of the quake epicenter?
  result <- try(stations <-
    getStation(
      iris,
      "US",
      "*",
```

```
      "*",
      "BHZ",
      starttime,
      endtime,
      latitude = e$latitude,
      longitude = e$longitude,
      maxradius = 5
    )
  )
  if (inherits(result,"try-error")) {
    message(geterrmessage())
 } else {
    stations
 }
}
```

---

getTimeseries *Retrieve seismic data from EarthScope*

Description

The getTimeseries method makes a request of the EarthScope timeseries webservice and returns a Stream object in which individual Traces have been sorted by start time.

Usage

```
getTimeseries(obj, network, station, location, channel,
              starttime, endtime,...)
```

Arguments

obj	IrisClient object
network	character string with the two letter seismic network code
station	character string with the station code
location	character string with the location code
channel	character string with the three letter channel code
starttime	POSIXct class specifying the starttime (GMT)
endtime	POSIXct class specifying the endtime (GMT)
...	optional arguments repository optional character string identifying whether to exclusively search primary archive or realtime collection buffers. Acceptable values are "primary" or "realtime". If not specified, IRIS webservices defaults to both repositories. processing optional character string with processing commands inclusiveEnd optional logical determining whether the endtime is inclusive (default = TRUE) ignoreEpoch optional logical defining behavior when multiple epochs are encountered (default = FALSE)

Details

This is an alternative method for retreiving seismic data that accepts optional processing commands. Data requests are made through the `timeseries` webservice and returned data are parsed using the internal miniseed2Stream() function.

If the location argument contains an empty string to specify a 'blank' location code, a location code of `"--"` will be used in the dataselect request URL.

The `processing` parameter can be used to specify any type of processing supported by the `timeseries` webs service. This string must begin with an ampersand and be ready to be appended to the request url, e.g. `processing="&demean=true&taper0.2,HANNING"`. This gives the user complete control over the number and order of processing commands. (See timeseries documentation.)

If `inclusiveEnd=FALSE`, then `getDataselect` will subtract 0.000001 seconds from the endtime before passing the value to the `dataselect` webservice. An endtime of, e.g., `as.POSIXct("2016-01-03", tz="GMT")` will be passed into `dataselect` as `end=2016-01-02T23:59:59.999999`. A data sample at time 2016-01-03T00:00:00 will not be returned unless `inclusiveEnd=TRUE`.

Sometimes the station webservice will return multiple records for the same SNCL, each with a different scale or starttime. These represent different *epochs* with potentially different metadata parameters for the SNCL and, by default, will cause a 'Multiple epochs' error message to be generated.

Handling all possible metadata differences so that the data may be merged is beyond the scope of this package. Instead, to avoid errors, users may specify `ignoreEpoch=TRUE` in which case the very first SNCL-epoch encountered will be used and all others will be discarded.

For access to restricted data, getDataselect will look for a system environmental variable "IrisClient_passw" which should be a string in the form "database_login:database_password". Alternatively,getDataselect will look for system environmental variable "IrisClient_netrc" which should point to a .netrc authentication file.

Error returns from the webservice will stop evaluation and generate an error message.

Value

A new `Stream` object is returned.

Author(s)

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

References

The EarthScope timeseries webservice:

https://service.earthscope.org/irisws/timeseries/1/

See Also

`getDataselect`, `getSNCL`, `IrisClient-class`

Examples

```
## Not run:
# Open a connection to EarthScope webservices (use debug=TRUE so we can see the URLs generated)
iris <- new("IrisClient",debug=TRUE)

starttime <- as.POSIXct("2013-06-01",tz="GMT")
endtime <- starttime + 24*3600

# Get raw data and processed data
st1 <- getDataselect(iris,"IU","ANMO","00","LHZ",starttime,endtime)
st2 <- getTimeseries(iris,"IU","ANMO","00","LHZ",starttime,endtime,"&demean=true&taper=0.2,HANNING")

layout(matrix(seq(2)))
plot(st1)
plot(st2)

## End(Not run)
```

---

getTraveltime *Retrieve seismic traveltime information from EarthScope*

Description

Deprecated due to the upcoming retirement of the EarthScope `traveltime` web service. Please use `taupTraveltime` instead.

The `getTraveltime` method obtains seismic traveltime data from the EarthScope `traveltime` web service and returns it in a dataframe.

Usage

```
getTraveltime(obj, latitude, longitude, depth, staLatitude, staLongitude)
```

Arguments

`obj`	an `IrisClient` object
`latitude`	latitude of seismic event
`longitude`	longitude of seismic event
`depth`	depth of seismic event
`staLatitude`	latitude of seismic station
`staLongitude`	longitude of seismic station

Details

The `traveltime` web service calculates travel-times for seismic phases using a 1-D spherical earth model.

Value

A dataframe with the following columns:

`distance, depth, phaseName, travelTime, rayParam, takeoff, incident`
`puristDistance, puristName`

Rows are ordered by `travelTime`.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope traveltime web service:

https://service.earthscope.org/irisws/traveltime/1/

See Also

IrisClient-class

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Two days around the "Nisqually Quake"
starttime <- as.POSIXct("2001-02-27", tz="GMT")
endtime <- starttime + 3600 * 24 * 2

# Find biggest seismic event over these two days -- it's the "Nisqually"
events <- getEvent(iris, starttime, endtime, minmag=5.0)
bigOneIndex <- which(events$magnitude == max(events$magnitude))
bigOne <- events[bigOneIndex[1],]

# Find US stations that are available within an hour of the event
start <- bigOne$time
end <- start + 3600
availability <- getChannel(iris, "US", "", "", "BHZ",
                           starttime=start, endtime=end,
                           latitude=bigOne$latitude, longitude=bigOne$longitude,
                           minradius=0, maxradius=10)

# Get the station the furthest East
minLonIndex <- which(availability$longitude == max(availability$longitude))
snclE <- availability[minLonIndex,]

# Plot the BHZ signal from this station
st <- getDataselect(iris,snclE$network,snclE$station,snclE$location,snclE$channel,
                    start,end)
```

```
# Check that there is only a single trace and then plot it
length(st@traces)
tr <- st@traces[[1]]
plot(tr, subsampling=1) # need subsmpling=1 to add vertical lines with abline()

# Find travel times to this station
traveltimes <- getTraveltime(iris, bigOne$latitude, bigOne$longitude, bigOne$depth,
                             snclE$latitude, snclE$longitude)

# Look at the list
traveltimes

# mark the P and S arrival times
pArrival <- start + traveltimes$travelTime[traveltimes$phaseName=="P"]
sArrival <- start + traveltimes$travelTime[traveltimes$phaseName=="S"]
abline(v=pArrival, col='red')
abline(v=sArrival, col='blue')

## End(Not run)
```

`getUnavailability` *Retrieve Channel metadata from EarthScope*

Description

This method is deprecated. It relies on the EarthScope fdsnws/station web service parameter "matchtimeseries" which is no longer supported. Without "matchtimeseries" this method will always return an empty dataframe.

The `getUnavailability` method obtains metadata for channels that are not available from the EarthScope station web service and returns it in a dataframe.

Usage

```
getUnavailability(obj, network, station, location, channel,
                  starttime, endtime, includerestricted,
                  latitude, longitude, minradius, maxradius)
```

Arguments

`obj`	`IrisClient` object
`network`	character string with the two letter seismic network code
`station`	character string with the station code
`location`	character string with the location code
`channel`	character string with the three letter channel code
`starttime`	POSIXct class specifying the starttime (GMT)
`endtime`	POSIXct class specifying the endtime (GMT)

includerestricted
: optional logical identifying whether to report on restricted data (default=FALSE)

latitude
: optional latitude used when specifying a location and radius

longitude
: optional longitude used when specifying a location and radius

minradius
: optional minimum radius used when specifying a location and radius

maxradius
: optional maximum radius used when specifying a location and radius

Details

The getUnavailability method compares the results of the getAvailability and getChannel methods and returns those records found only in the output of getChannel.

Each of the arguments network, station, location or channel may contain a valid code or a wildcard expression, e.g. "BH?" or "*". Empty strings are converted to "*". Otherwise the ascii string that is used for these values is simply inserted into the web service request URL.

For more details see the webservice documentation.

Value

A dataframe with the following columns:

```
network, station, location, channel, latitude, longitude, elevation, depth,
  azimuth, dip, instrument, scale, scalefreq, scaleunits, samplerate,
  starttime, endtime
```

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

The EarthScope station webservice:

https://service.earthscope.org/fdsnws/station/1/

This implementation was inspired by the functionality in the obspy get_stations() method.

https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.get_stations.html

See Also

IrisClient-class, getAvailability, getChannel

Examples

```
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Date of Nisqually quake
starttime <- as.POSIXct("2001-02-28",tz="GMT")
endtime <- starttime + 2*24*3600
```

```
# Use the getEvent web service to determine what events happened in this time period
result <- try(events <- getEvent(iris,starttime,endtime,6.0))
if (inherits(result,"try-error")) {
  message(geterrmessage())
} else {
  events

# biggest event is Nisqually
  eIndex <- which(events$magnitude == max(events$magnitude))
  e <- events[eIndex[1],]

# Find all BHZ channels that were NOT collecting data at the time of the quake
# and within 5 degrees of the quake epicenter (or are otherwise unavailable from IRIS).
  result <- try(channels <- getUnavailability(iris,"*","*","*","BHZ",starttime,endtime,
                            lat=e$latitude,long=e$longitude,maxradius=5))
  if (inherits(result,"try-error")) {
    message(geterrmessage())
  } else {
    channels
  }
}
```

`getUpDownTimes` *Determine times when a channel starts/stops collecting data*

Description

The `getUpDownTimes` method determines the on/off times for data collection within a `Stream` and returns a list containing these times, ignoring `Traces` with a duration less than `min_signal` as well as data dropouts that are less than `min_gap`.

Usage

```
getUpDownTimes(x, min_signal, min_gap)
```

Arguments

`x`	`Stream` object
`min_signal`	minimum `Trace` duration in seconds (default=30)
`min_gap`	minimum gap in seconds (default=60)

Details

A `Stream` object returned by `getDataselect` contains a list of individual `Trace` objects, each of which is guaranteed to contain a continuous array of data in the `Trace@data` slot. Each `Trace` also contains a `starttime` and an `endtime` representing a period of uninterrupted data collection. Data dropouts are determined by first rejecting any `Traces` of duration less than `min_signal`. The temporal spacing between `Traces` is then analyzed, ignoring spaces shorter than `min_gap`.

This method first checks the SNCL id of each Trace to make sure they are identical and throws an error if they are not.

The first element returned is always the starttime associated the first Trace. The last element is always the endtime associated with the last trace. Thus, when the first element is identical to the starttime of the web services data request this does not necessarily mean that the channel was down before this.

NOTE: Even when data are complete for the duration of the requested timespan, the last element returned may be earlier than the endtime of the web services data request by up to a second.

Value

A vector of POSIXct datetimes associated with on/off transitions.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

plotUpDownTimes

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Determine up/down transitions, ignoring Traces < 3 min and gaps < 5 min
upDownTimes <- getUpDownTimes(st, min_signal=180, min_gap=300)

# Or just plot them directly
plotUpDownTimes(st, min_signal=180, min_gap=300)

## End(Not run)
```

---

hilbert *Hilbert of a seismic signal*

Description

The hilbert method of Trace objects returns a Trace whose data have been replaced with the Hilbert transform of the seismic signal.

Usage

```
hilbert(x)
```

Arguments

x	a Trace object

Details

Before calculating the Hilbert transform, the seismic trace is 'cleaned up' by removing the mean, the trend and by applying a cosine taper. See DDT for more details.

Value

A Trace whose data have been replaced with the Hilbert transform of the seismic signal.

Note

This algorithm is adapted from code in the **seewave** package.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2010-02-27 06:00:00", tz="GMT")
endtime <- as.POSIXct("2010-02-27 09:00:00", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]

# Create Hilbert transform of the trace
trh <- hilbert(tr)

# Plot signal data and hilbert data
plot(tr@data, type='l', col='gray80')
points(trh@data, type='l', col='blue')

## End(Not run)
```

hilbertFFT *Hilbert FFT*

Description

The hilbertFFT function returns the complex Hilbert FFT of a timeseries signal.

Usage

```
hilbertFFT(x)
```

Arguments

x	a numeric vector

Details

This function is intended for internal use by the hilbert() and envelope() methods of Trace objects.

Value

A complex vector containing the Hilbert FFT of x.

Note

This algorithm is adapted from code in the **seewave** package.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2010-02-27 06:00:00", tz="GMT")
endtime <- as.POSIXct("2010-02-27 09:00:00", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]

# Demean, detrend, cosine taper
tr <- DDT(tr)

# Calculate Hilbert FFT of the trace data
hfft <- hilbertFFT(tr@data)
```

```
# Plot signal, with Hilbert envelope
layout(1)
plot(tr@data, type='l', col='gray80', main="Signal and Envelope")
points(Mod(hfft), type='l', col='blue')

# 2 rows
layout(matrix(seq(2)))

# Show that Imaginary component of Hilbert transform has the
# original signal shifted by 90 degrees
ccf(tr@data,tr@data,lag.max=200,main="Auto-correlation of signal data")
ccf(tr@data,Im(hfft),lag.max=200,main="90 deg phase shift with Hilber transform")

# Restore default layout
layout(1)

## End(Not run)
```

`IrisClient-class` *Class* `"IrisClient"`

Description

A class for making data and metadata requests from EarthScope web services.

Slots

`site`**:** Object of class `"character"`: this is the top level domain that the web services will be pointed to; defaults to "https://service.earthscope.org"

`service_type`**:** Object of class `"character"`: defaults to `fdsnws`; for data retrieval from the IRIS PH5 repository, set this to `ph5ws`;

`debug`**:** Object of class `"logical"`: when set to TRUE will cause any web service requestURL to be printed

`useragent`**:** Object of class `"character"`: client identification string

`event_site`**:** Object of class `"character"`: this is the top level domain that the event web service url will be pointed to; defaults to `site` or if `site="https://service.earthscope.org"` or `site="https://service.iris.edu"`, defaults to `"https://earthquake.usgs.gov"`

`dataselect_site`**:** Object of class `"character"`: this is the top level domain that the dataselect web service url will be pointed to; defaults to `site`

`station_site`**:** Object of class `"character"`: this is the top level domain that the station web service url will be pointed to; defaults to `site`

`availability_site`**:** Object of class `"character"`: this is the top level domain that the availability web service url will be pointed to; defaults to `site`

`retries`**:** Object of class `"numeric"`: set the number of retries for API calls when receiving a server error response, must be 5 or fewer

Methods

getAvailability Deprecated. Makes a channel request of the station web service and returns the result as a dataframe; see `getAvailability`

getChannel makes a channel request of the station web service and returns the result as a dataframe; see `getChannel`

getDataAvailability makes a channel request of the irisws availability web service and returns the result as a dataframe; see `getDataAvailability`

getDataselect makes a request of the dataselect web service and returns a `Stream` object; see `getDataselect`

getDistaz makes a request of the distaz web service and returns a the information as a dataframe; see `getDistaz`

getEvalresp makes a request of the instrument response web service and returns the information as a dataframe; see `getEvalresp`

getEvent makes a request of the event web service and returns the information as a dataframe; `getEvent`

getNetwork makes a network request of the station web service and returns the result as a dataframe; see `getNetwork`

getSNCL: calls the `getDataselect` method and returns a `Stream` object; see `getSNCL`

getStation makes a station request of the station web service and returns the result as a dataframe; see `getStation`

getTraveltime makes a request of the traveltime web service and returns the information as a dataframe; see `getTraveltime`

getUnavailability Deprecated. Makes a channel request of the station web service and returns the result as a dataframe; see `getUnavailability`

Note

The `IrisClient` object is inspired by the `clients.fdsn.client.Client` class found in the python ObsPy package (https://docs.obspy.org/packages/autogen/obspy.clients.fdsn.client.Client.html).

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)
```

```
mean(st)

## End(Not run)
```

`IRISSeismic-deprecated` *Deprecated Functions in Package 'IRISSeismic'*

Description

These functions are deprecated and may be defunct as soon as the next release.

- `getTraveltime.IrisClient`: getTraveltime has a dependency on the EarthScope web service 'traveltime' which will not be supported in the future.
- `getDistaz.IrisClient`: getDistaz has a dependency on the EarthScope web service 'distaz' which will not be supported in the future.
- `getAvailability.IrisClient`: getAvailability has a dependency on the EarthScope station web service parameters 'matchtimeseries' and 'includeavailability' which are no longer available. Without these parameters, getAvailability returns the same result as `getChannel.IrisClient`.
- `getUnavailability.IrisClient`: getUnavailability has a dependency on the EarthScope station web service parameters 'matchtimeseries' and 'includeavailability' which are no longer available. Without these parameters, getUnavailability returns an empty dataframe.

`McNamaraBins` *McNamara Binning*

Description

The McNamaraBins() function implements the binning algorithm specified in the "Data Preparation and Processing" section of Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions.

Usage

```
McNamaraBins(df, loFreq, hiFreq, alignFreq)
```

Arguments

`df`	an R `dataframe` object
`loFreq`	optional lo end of frequency binning range (default=.005)
`hiFreq`	optional hi end of frequency binning range (default=10)
`alignFreq`	optional alignment frequency for determining frequency bins (default=0.1)

Details

The McNamaraBins() function accepts a dataframe with an arbitrary number of columns. At least one of the columns must be named 'freq' and must contain frequency values. These frequencies will be used to assign all associated values into appropriate bins according to the McNamara algorithm:

Frequencies for binning are generated at 1/8 octave intervals aligned to `alignFreq`. Binned values associated with each frequency bin are calculated by averaging incoming values over an entire octave centered on that frequency.

Value

A dataframe containing binned values is returned with the same column names as the incoming `df` argument.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD

McNamaraPSD *Power Spectral Density*

Description

The McNamaraPSD() function implements the spectral density algorithm specified in the "Data Preparation and Processing" section of Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions.

Usage

```
McNamaraPSD(tr, loFreq=.005, hiFreq=10, alignFreq=0.1, binned=TRUE)
```

Arguments

`tr`	a Trace object
`loFreq`	optional lo end of frequency binning range
`hiFreq`	optional hi end of frequency binning range
`alignFreq`	optional alignment frequency for determining frequency bins
`binned`	logical determining whether the return spectrum is binned

Details

This PSD algorithm is designed to be used on one to three hour segments of seismic data and will return a PSD object containing the (potentially binned) spectrum for that segment. See the psdList function for automatic segmenting of longer Stream objects.

The McNamara PSD algorithm is similar to MATLAB's pwelch() function and has the following steps:

1. Calculate averaged spectrum

```
# Truncate incoming segment of trace data to nearest power of 2 samples.
# Divide each truncated segment into 13 chunks with 75% overlap. The first
# chunk begins at 0/16 and ends at 4/16.  The 13'th chunk begins at 12/16
# and ends at 16/16.  The chunks overlap like this:
#
#   1---5---9---3---
#    2---6---0---
#     3---7---1---
#      4---8---2---
#
# Deman, detrend and taper the chunk.
# Calculate the 'one-sided' spectrum for the chunk.
#
# Average together all 13 spectra to get an averaged spectrum.
```

2. Create smoothed version of spectrum with binning

   When binned=TRUE, McNamara style binning is turned on and a smoothed spectrum is returned that contains many fewer points than the full spectrum. When these arguments are not specified, binning is automatically turned off and the full spectrum is returned.

   Frequencies for binning are generated at 1/8 octave intervals aligned to alignFreq. The power (dB) associated with each frequency bin is calculated by averaging over an entire octave centered on that frequency.

   Note: The spectra returned by McNamaraPSD() have not had instrument correction applied. Use getEvalresp to get instrument correction values for specific frequencies.

3. convert binned spectra to decibels

Value

An R list object with the following named elements:

```
freq, spec, snclq, starttime, endtime
```

Elements freq and spec are numeric vectors while snclq, starttime and endtime are single values.

Note

During the binning process, an arithmetic mean is used to average together power levels in decibels. This is equivalent to averaging of power levels before conversion to dB using a geometric mean.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraBins, psdList

---

mergeTraces *Merge multiple traces into a single trace*

Description

The mergeTraces method of Stream objects returns a new Stream where all Traces have been merged into a single Trace. Gaps between traces are replaced with values determined by the fillMethod parameter.

Usage

```
mergeTraces(x, fillMethod)
```

Arguments

x	Stream object
fillMethod	method to use when filling gaps between Traces (default="fillNA")

Details

Available values for fillMethod include:

- fillNA – gaps are filled with NA (R's missing value flag)
- fillZero – gaps are filled with 0.0

Value

A new Stream object containing a single Trace is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2002-04-20", tz="GMT")
endtime <- as.POSIXct("2002-04-21", tz="GMT")
st4 <- getDataselect(iris,"US","OXF","","BHZ",starttime,endtime)
stm4 <- mergeTraces(st4)

# plot merged trace
plot(stm4@traces[[1]])
mtext(paste(length(st4@traces),"traces"), side=3, line=0.5, adj=0.05, cex=1.5)

## End(Not run)
```

---

mergeUpDownTimes *Determine overlaps in two sets of upDownTimes.*

Description

The mergeUpDownTimes function determines the overlaps in two sets of times representing up/down (on/off) periods for a single or a set of channels. This function can be used to determine overall station up/down periods.

Usage

```
mergeUpDownTimes(udt1, udt2, bothOn)
```

Arguments

udt1	vector of POSIXct times representing up/down transitions
udt2	vector of POSIXct times representing up/down transitions
bothOn	logical specifying whether overlaps are determined with AND or OR (default=FALSE: udt1 OR udt2)

Details

When bothOn=FALSE, the default, this function returns the times of transitions from "either to neither" and back. When bothOn=TRUE, this function returns the times of transitions from "both to either" and back.

If an empty vector is passed in for udt1 or udt2 then the other vector is returned unchanged. This can be useful when merging the upDownTimes for multiple channels. See the example below.

Value

A vector of POSIXct datetimes associated with on/off transitions.

Note

The vector of times in `udt1` and `udt2` has no information on the values of `min_signal` or `min_gap` that were used to generate the timeseries. It is up to the user to make sure that the incoming vectors are appropriate for comparison. See `getUpDownTimes`.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

`getUpDownTimes`, `plotUpDownTimes`

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Three Streams, each with different upDownTimes
starttime <- as.POSIXct("2012-07-01", tz="GMT")
endtime <- as.POSIXct("2012-07-02", tz="GMT")
stE <- getDataselect(iris,"IU","XMAS","10","BHE",starttime,endtime)
stN <- getDataselect(iris,"IU","XMAS","10","BHN",starttime,endtime)
stZ <- getDataselect(iris,"IU","XMAS","10","BHZ",starttime,endtime)
udtE <- getUpDownTimes(stE)
udtN <- getUpDownTimes(stN)
udtZ <- getUpDownTimes(stZ)

udtAll <- c()
udtAny <- c()
for (udt in list(udtE, udtN, udtZ)) {
  udtAll <- mergeUpDownTimes(udtAll,udt,bothOn=TRUE)
  udtAny <- mergeUpDownTimes(udtAny,udt,bothOn=FALSE)
}

# 5 rows
layout(matrix(seq(5)))

# Plot the results
par(mar=c(3,4,3,2)) # adjust margins
plotUpDownTimes(udtE); title("BHE")
plotUpDownTimes(udtN); title("BHN")
plotUpDownTimes(udtZ); title("BHZ")
plotUpDownTimes(udtAll); title("ALL channels up")
plotUpDownTimes(udtAny); title("ANY channel up")

# Restore default layout
layout(1)

## End(Not run)
```

`miniseed2Stream` *Convert miniSEED bytes to* `Stream` *object*

Description

The `miniseed2Stream` function converts raw miniSEED bytes into a `Stream` object.

Usage

```
miniseed2Stream(miniseed, url, requestedStarttime, requestedEndtime,
                sensor, scale, scalefreq, scaleunits, latitude, longitude,
                elevation, depth, azimuth, dip, removeZeroSampleTraces)
```

Arguments

`miniseed`	a vector of raw bytes read from a miniSEED file
`url`	optional, character source location (see getDataselect)
`requestedStarttime`	optional, `POSIXct` time associated with the requested starttime (see getDataselect)
`requestedEndtime`	optional, `POSIXct` time associated with the requested endtime (see getDataselect)
`sensor`	optional, character description of the Sensor type associated with this Station-Network-Channel-Location (SNCL) (see Trace)
`scale`	optional, character description of the InstrumentSensitivity associated with this SNCL (see Trace)
`scalefreq`	optional, numeric description of frequency at which the InstrumentSensitivity is correct, the SensitivityFrequency (see Trace)
`scaleunits`	optional, character description of the InputUnits associated with this SNCL (see Trace)
`latitude`	optional, numeric latitude associated with this SNCL (see Trace)
`longitude`	optional, numeric longitude associated with this SNCL (see Trace)
`elevation`	optional, numeric elevation associated with this SNCL (see Trace)
`depth`	optional, numeric depth associated with this SNCL (see Trace)
`azimuth`	optional, numeric channel azimuth associated with this SNCL (see Trace)
`dip`	optional, numeric channel dip associated with this SNCL (see Trace)
`removeZeroSampleTraces`	optional, logical determining whether traces with no sample values are removed (default = TRUE)

Details

This function takes raw bytes read in from a file or URL and converts them to a `Stream` object. Metadata information is optional. This function is primarily for internal use.

Value

A Stream object.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

readMiniseedFile

---

multiplyBy *Multiplication by a constant*

Description

The multiplyBy methods of Trace and Stream objects return like objects where all @data slots have been multiplied by a constant.

Usage

```
multiplyBy(x, y)
```

Arguments

x	a Trace or Stream object
y	a numeric multiplier

Value

A new Trace or Stream object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2011-01-24", tz="GMT")
endtime <- as.POSIXct("2011-01-25", tz="GMT")

# Get the waveform
stRaw <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)
```

```
# obtain an instrument sensitivity value with getChannel metadata)
c <- getChannel(iris, "AK","PIN","","BHZ",starttime, endtime)
sensitivityValue <- c$scale

# convert raw data
st <- multiplyBy(stRaw, 1/sensitivityValue)
rmsVariance(st)

# plot trace
plot(st, ylab=c$scaleunits)

## End(Not run)
```

`noiseMatrix2PdfMatrix` *Convert matrix of PSDs to matrix of Probability Density Functions*

Description

This function converts a `noiseMatrix` returned by either `psdList2NoiseMatrix` or `psdDF2NoiseMatrix` into a matrix of Probability Density values as defined by McNamara and Boaz 2005.

Usage

```
noiseMatrix2PdfMatrix(noiseMatrix, lo, hi, binSize)
```

Arguments

`noiseMatrix`	a `noiseMatrx` returned from either `psdList2NoiseMatrix` or `psdDF2NoiseMatrix`
`lo`	lowest frequency bin (power level in dB) for the PDF y-axis (default=-200)
`hi`	highest frequency bin (power level in dB) for the PDF y-axis (default=-50)
`binSize`	size in dB of each bin (default=1)

Details

The McNamara and Boaz paper describes creating histograms of the discretized power levels at each frequency bin associated with a set of PSDs. The value in each cell of the PDF matrix is the fraction of the corrected PSDs that have that power level at that frequency bin.

To return a PDF matrix that matches those in the McNamara paper, use the default settings.

Value

A `matrix` is returned with one row for each power level (-250:-50 dB) and one column for each frequency bin.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdDF2NoiseMatrix, psdList, psdPlot, psdStatistics

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT") # 2011.125
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)

# Generate power spectral density for each hour long segment
psdList <- psdList(st)

# Convert into corrected "noiseMatrix"
noiseMatrix <- psdList2NoiseMatrix(psdList)

# Convert into McNamara "pdfMatrix"
pdfMatrix <- noiseMatrix2PdfMatrix(noiseMatrix)

# NOTE:  Data need to be flipped and tranposed for the XY axes in the
# NOTE:  image() function to match rows and columns in our pdfMatrix
# Plot pdfMatrix
image(t(pdfMatrix[,ncol(pdfMatrix):1]),
      col=c('gray90',rainbow(9)),
      axes=FALSE)

## End(Not run)
```

---

noiseModels *Generate NHNM and NLNM noise models*

Description

The noiseModels function returns the New High Noise Model and New Low Noise Model from the Peterson paper referenced below. Values are returned for the specific frequencies specified in the freq argument.

Usage

```
noiseModels(freq)
```

Arguments

freq	a vector of frequencies at which to generate noise model values

Value

A list is returned with elements nhnm and nlnm containing the high and low noise models, respectively.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Observations of Modeling and Seismic Background Noise (Peterson 1993) Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

psdStatistics,

---

psdDF2NoiseMatrix *Apply instrument correction to PSDs*

Description

The psdDF2NoiseMatrix function uses the snclq identifier associated with the first PSD in the dataframe to obtain instrument correction information at the specified frequencies from the getEvalresp web service if instrumentation correction information is not supplied as an argument. This correction is applied to every PSD in the dataframe and the now corrected PSD values are returned as a matrix.

Usage

```
psdDF2NoiseMatrix(DF, evalresp=NULL)
```

Arguments

DF	a dataframe of PSDs obtained from the getPSDMeasurements method of IrisClient.
evalresp	dataframe of freq, amp, phase information matching output of getEvalresp, optional.

Details

This function is identical in behavior to psdList2NoiseMatrix except that the input object is a dataframe of PSD values obtained from the MUSTANG Backend Storage System.

Value

A matrix is returned with one row for each instrument-corrected PSD and one column for each frequency bin.

Note

The incoming dataframe is checked to make sure that it represents only a single SNCL (Station-Network-Channel-Location). An error is generated if more than one is found. However, the psdDF is not checked to make sure that no changes to the instrument correction happened during the time period covered by the psdDF. This occurs at an 'epoch' boundary when an instrument is replaced.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdList, psdList2NoiseMatrix, psdPlot, psdStatistics

---

psdList *Apply McNamara PSD algorithm to a seismic signal*

Description

The psdList function subsets a seismic Stream object into a series of shorter segments with 50% overlap and uses the McNamaraPSD method to return a smoothed (aka binned) Power Spectral Density (PSD) for each segment.

Usage

```
psdList(st)
```

Arguments

st	a Stream object

Details

A Stream will be subset into segments depending upon the channel identifier (@stats@channel) associated with this seismic data. The binning frequencies are also channel dependent as exemplified in this code extract where Z is the segment length in seconds:

```
alignFreq <- 0.1

if (stringr::str_detect(channel,"^L")) {
  Z <- 3 * 3600
  loFreq <- 0.001
  hiFreq <- 0.5 * tr_merged@stats@sampling_rate
} else if (stringr::str_detect(channel,"^M")) {
  Z <- 2 * 3600
  loFreq <- 0.0025
  hiFreq <- 0.5 * tr_merged@stats@sampling_rate
} else {
  Z <- 3600
  loFreq <- 0.005
  hiFreq <- 0.5 * tr_merged@stats@sampling_rate
}
```

Each new segment starts half way through the previous segment. (50% overlap)

Value

A list of PSD objects is returned. Each element of the list is an R list object with the following elements:

```
freq, spec, snclq, starttime, endtime
```

Note: Individual PSDs have not had instrument correction applied.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdList2NoiseMatrix, psdPlot, psdStatistics,

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT") # 2011.125
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)

# Generate power spectral density for each hour long segment
psdList <- psdList(st)

# Plot uncorrected PSDs
period <- 1/psdList[[1]]$freq
plot(period, psdList[[1]]$spec, log='x', type='l',
     xlab="Period (Sec)", ylab="Power (dB)",
     main="Uncorrected PSDs")

for (i in seq(2:length(psdList))) {
  points(period, psdList[[i]]$spec, type='l')
}

## End(Not run)
```

`psdList2NoiseMatrix` *Apply instrument correction to PSDs*

Description

The `psdList2NoiseMatrix` function uses the `snclq` identifier associated with the first PSD in the list to obtain instrument correction information at the specified frequencies from the `getEvalresp` web service if instrumentation correction information is not supplied as an argument. This correction is applied to every PSD in the list and the now corrected PSD values are returned as a matrix.

Usage

```
psdList2NoiseMatrix(psdList, evalresp=NULL)
```

Arguments

`psdList`	a list of PSDs generated by the `psdList` function
`evalresp`	dataframe of freq, amp, phase information matching output of `getEvalresp`, optional

Value

A `matrix` is returned with one row for each instrument-corrected PSD and one column for each frequency bin.

Note

The psdList function generates a psdList from a single Stream of data and should thus only contain data for a single SNCL (Station-Network-Channel-Location). However, the psdList is not checked to make sure that no changes to the instrument correction happened during the time period covered by the psdList. This occurs at an 'epoch' boundary when an instrument is replaced.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdDF2NoiseMatrix, psdList, psdPlot, psdStatistics,

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT") # 2011.125
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)

# Generate power spectral density for each hour long segment
psdList <- psdList(st)

# Convert into corrected "noiseMatrix"
noiseMatrix <- psdList2NoiseMatrix(psdList)

# Plot corrected PSDs
period <- 1/psdList[[1]]$freq
plot(period, noiseMatrix[1,], log='x', type='l',
     ylim=c(-200,-50),
     xlab="Period (Sec)", ylab="Power (dB)",
     main="Corrected PSDs")

for (i in seq(2:nrow(noiseMatrix))) {
  points(period, noiseMatrix[i,], type='l')
}

## End(Not run)
```

psdPlot *Generate plots from a set of PSDs*

Description

The psdPlot function is used to generate plots from the data in a psdList or psdDF dataframe.

Usage

```
psdPlot(PSDs, style='psd', evalresp=NULL, ylo=-200, yhi=-50, showNoiseModel=TRUE,
        showMaxMin=TRUE, showMode=TRUE, showMean=FALSE, showMedian=FALSE, ...)
```

Arguments

PSDs	either a list as returned by psdList or a dataframe of PSD values obtained from the BSS
style	character identifier of plot type: 'psd' plots PSD lines, 'pdf' plots the pdfMatrix
evalresp	dataframe of freq, amp, phase information matching output of getEvalresp, optional
ylo	numeric setting lower limit of plot y-axis (default=-200)
yhi	numeric setting upper limit of plot y-axis (default=-50)
showNoiseModel	logical controlling plotting of noise model lines (default=TRUE)
showMaxMin	logical controlling plotting of PSD max and min lines (default=TRUE)
showMode	logical controlling plotting of PDF mode line (default=TRUE)
showMean	logical controlling plotting of PSD mean line (default=FALSE)
showMedian	logical controlling plotting of PSD median line (default=FALSE)
...	arguments to be passed to plotting methods

Details

The psdPlot function creates visualizations for sets of PSDs. Plots generated with style='pdf' mimic the plots presented in the McNamara paper.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdList, psdStatistics

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT") # 2011.125
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)

# Generate power spectral density for each hour long segment
psdList <- psdList(st)

# 'psd' line plot
psdPlot(psdList,style='psd',type='l',col=adjustcolor('black',0.3))

# McNamara 'pdf' plot
psdPlot(psdList,style='pdf')

## End(Not run)
```

---

psdStatistics *Return statistics for a set of PSDs*

Description

The psdStatistics function calculates a variety of information associated with the incoming set of PSDs.

Usage

```
psdStatistics(PSDs, evalresp=NULL)
```

Arguments

PSDs	either a list as returned by psdList or a dataframe of PSD values obtained from the BSS
evalresp	dataframe of freq, amp, phase information matching output of getEvalresp, optional

Value

A list of elements:

- noiseMatrix – matrix of corrected power levels; rows=PSDs, columns=frequencies
- pdfMatrix – matrix of probability density values; rows=dB level, columns=frequencies
- freq – vector of frequencies associated statistics vectors and with matrix columns

- pdfBins – vector of power values (dB) associated with pdfMatrix rows
- max – maximum power level at each frequency
- min – minimum power level at each frequency
- mean – mean power level at each frequency
- median – median power level at each frequency
- mode – mode of power level at each frequency (obtained from pdfMatrix)
- nlnm – low noise model power level at each frequency
- nhnm – high noise model power level at each frequency
- pct_above – percent of PSDs above the high noise model at each frequency
- pct_below – percent of PSDS below the low noise model at each frequency

A variety of plots can be generated form the information in this list.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

Seismic Noise Analysis System Using Power Spectral Density Probability Density Functions (McNamara and Boaz 2005)

See Also

McNamaraPSD, psdList, psdPlot

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT") # 2011.125
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)

# Generate power spectral density for each hour long segment
psdList <- psdList(st)

# Generate Statistics
stats <- psdStatistics(psdList)

# Just for fun plot
logPeriod <- log10(1/stats$freq)
plot(logPeriod,stats$max,ylim=c(-200,-50), las=1,
     xlab="log10(period)", ylab="Power (dB)",
     main="Model 'normal background noise' area and area of seismic signal.")
```

```
points(logPeriod,stats$min)

# Overlay a polygon showing the range between the noise models
x <- c(logPeriod,rev(logPeriod),logPeriod[1])
y <- c(stats$nhnm,rev(stats$nlnm),stats$nhnm[1])
transparentBlack <- adjustcolor('black',0.4)
polygon(x,y,col=transparentBlack)

# Overlay a polygon showing the range of measured values
y <- c(stats$max,rev(stats$min),stats$max[1])
transparentBlue <- adjustcolor('blue',0.6)
polygon(x,y,col=transparentBlue)

## End(Not run)
```

`readMiniseedFile` *Convert miniSEED file to* `Stream` *object*

Description

The `readMiniseedFile` function converts a raw miniSEED file into a `Stream` object.

Usage

```
readMiniseedFile(file,sensor,scale,scalefreq,scaleunits,
                 latitude,longitude,elevation,depth,azimuth,dip)
```

Arguments

`file`	character path of a miniSEED file
`sensor`	character description of the Sensor associated with this Station-Network-Channel-Location (SNCL) (see Trace)
`scale`	numeric description of the InstrumentSensitivity associated with this SNCL (see Trace)
`scalefreq`	numeric description of frequency at which the InstrumentSensitivity is correct, the SensitivityFrequency (see Trace)
`scaleunits`	character description of the InputUnits associated with this SNCL (see Trace)
`latitude`	numeric latitude associated with this SNCL (see Trace)
`longitude`	numeric longitude associated with this SNCL (see Trace)
`elevation`	numeric elevation associated with this SNCL (see Trace)
`depth`	numeric depth associated with this SNCL (see Trace)
`azimuth`	numeric channel azimuth associated with this SNCL (see Trace)
`dip`	numeric channel dip associated with this SNCL (see Trace)

Details

This function reads in a raw miniSEED file and converts it to a Stream object. Metadata information is optional.

Value

A Stream object.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

miniseed2Stream

---

rms *Root Mean Square and RMS Variance*

Description

The rms and rmsVariance methods of Trace and Stream objects compute the Root Mean Square (RMS) amplitude or RMS variance of the associated data in each object. RMS variance removes the DC level from the seismic signal so that the zero line is consistent.

Usage

```
rms(x, na.rm)
parallelRms(x, na.rm)
rmsVariance(x, na.rm)
parallelRmsVariance(x, na.rm)
```

Arguments

x	a Trace or Stream object
na.rm	a logical specifying whether missing values should be removed

Details

Trace method

The RMS amplitude of a single Trace is calculated as:

$$rms(x) = \sqrt{\frac{\sum_1^n (x_i)^2}{n}}$$

The RMS variance of a single Trace is calculated as:

$$rmsVariance(x) = \sqrt{\frac{\sum_1^n (x_i - \bar{x})^2}{n}}$$

where x is the vector of data values and n is the length of that vector.

Stream methods

For Stream objects, data from all Traces in the stream are first extracted and concatenated into a single numeric vector after which the algorithm is applied.

The parallel~ version of this method is only available on Stream objects and returns a vector of values, one for each Trace.

By default, the Stream versions of these methods use na.rm=FALSE as there should be no missing datapoints in each Trace. The Trace methods default to na.rm=TRUE to accommodate merged traces where gaps between traces have been filled with NAs.

Value

A single numeric value is returned or NA if the trace has no data.

A numeric vector is returned for parallelRmsVariance.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:

# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Get the first trace and generate some statistics
tr <- st@traces[[1]]
rmsVariance(tr)

## End(Not run)
```

rotate2D *Rotate horizontal components of a seismic signal*

Description

The rotate2D() function rotates the two horizontal components of a seismic signal into Radial and Transverse components returned as a list of 2 Stream objects.

Usage

```
rotate2D(st1, st2, angle)
```

Arguments

st1	horizontal Stream from a channel set (channel name usually ending in "N", "E", "1", or "2")
st2	horizontal Stream from a channel set, complementary to st1
angle	angle (degrees) of the rotation

Details

The rotation web service returns Radial and Transverse seismic Streams, generated by rotating st1 and st2 by angle degrees.

The rotation service uses the following transformation matrix to change the output vectors for 2-D horizontal transformations

$$M_{2D} = \begin{bmatrix} cos\alpha & sin\alpha \\ -sin\alpha & cos\alpha \end{bmatrix}$$

$$\begin{bmatrix} R \\ T \end{bmatrix} = M_{2D} \begin{bmatrix} N \\ E \end{bmatrix}$$

where :

```
N and E represent data from the original (horizontal) orientations.
R and T represent the Radial and Transverse components.
\eqn{\alpha} is the azimuth angle measured clockwise from north.
```

Value

A list of two Stream objects stR and stT is returned.

Note

N and E are determined by the Stream @stats@azimuth values. If Stream @stats@azimuth values are not defined, st1 is assumed to be N and st2 is assumed to be E. Orthogonality is also assumed to be correct.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

EarthScope rotation web service:

https://service.earthscope.org/irisws/rotation/1/

---

slice *Slice a section out of a* Trace *or* Stream

Description

The slice methods of Trace and Stream objects return like objects that are subsets of the original.

Usage

```
slice(x, starttime, endtime)
```

Arguments

x	a Trace or Stream object
starttime	time at which the slice should begin
endtime	time at which the slice should end

Details

The returned object will always be a subset of the x argument whose time range is the intersection of the original time range and the requested range. When there is no intersection or when starttime > endtime an error is generated.

All metadata associated with the returned Trace or Stream will reflect the new object, rather than the original.

Value

A new Trace or Stream object is returned.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2002-04-20", tz="GMT")
endtime <- as.POSIXct("2002-04-21", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"US","OXF","","BHZ",starttime,endtime)

# This Stream object consists of 5 Traces
length(st@traces)

# Plotting the third trace shows a small quake
plot(st@traces[[3]])

# We can slice out the hour that has the quake signal
sliceStart <- as.POSIXct("2002-04-20 10:30:00", tz="GMT")
sliceEnd <- as.POSIXct("2002-04-20 11:30:00", tz="GMT")
stSlice <- slice(st, sliceStart, sliceEnd)

# Now we only have one Trace of an hour duration
length(stSlice@traces)
stSlice@traces[[1]]@stats
# And a better look at the quake signal
plot(stSlice@traces[[1]])

## End(Not run)
```

STALTA *STA/LTA*

Description

The STALTA method of Trace objects applies one of several STA/LTA "first break picking" algorithms to Trace data in order to automatically detect seismic events.

Usage

```
STALTA(x, staSecs, ltaSecs, algorithm, demean, detrend, taper, increment)
```

Arguments

x	a Trace object
staSecs	length of the **S**hort averaging window in secs (default=3)
ltaSecs	length of the **L**ong averaging windowin secs (default=30)
algorithm	algorithm to be used (default="classic_LR")

demean	boolean flag determining whether to demean the data before applying the algorithm (default=TRUE)
detrend	boolean flag determining whether to detrend the data before applying the algorithm (default=TRUE)
taper	proportion of the signal to be tapered at each end before applying the algorithm (default=0.0)
increment	the increment to use when sliding the averaging windows to the next location (default=1).

Details

By default, this method uses the "classic_LR" algorithm which calculates the average power in the Trace data over a short window (STA) and a long window (LTA). With this algorithm, windows are "left/right aligned" meaning that the point for which STA/LTA is calculated is at the lefttmost edge of the STA window and the rightmost edge of the LTA window. The resulting STA/LTA ratio thus has the same number of points as the original data. This is a standard method of "first break picking" and can be used to identify the onset of a seismic event.

Three different algorithms are currently available:

1) algorithm="classic_RR" This is the original STA/LTA algorithm with "right alignment".

$$STA(x_i) = \frac{1}{ns} \sum_{j=i-ns}^{i} x_i^2$$

$$LTA(x_i) = \frac{1}{nl} \sum_{j=i-nl}^{i} x_i^2$$

$$r_i = \frac{STA_i}{LTA_i}$$

```
[---------- LTA ---------*]
                [-- STA -*]
```

2) algorithm="classic_LR" (default) This algorithm has the index at the left edge of the STA window and the right edge of the LTA window

$$STA(x_i) = \frac{1}{ns} \sum_{j=i}^{i+ns} x_i^2$$

$$LTA(x_i) = \frac{1}{nl} \sum_{j=i-nl}^{i} x_i^2$$

$$r_i = \frac{STA_i}{LTA_i}$$

```
[---------- LTA --------*]
                       [*- STA --]
```

3) `algorithm="EarleAndShearer_envelope"`

$$STA(x_i) = \frac{1}{ns} \sum_{j=i}^{i+ns} Mod(H(x))_i$$

$$LTA(x_i) = \frac{1}{nl} \sum_{j=i-nl}^{i} Mod(H(x))_i$$

$$r_i = \frac{STA_i}{LTA_i}$$

```
[---------- LTA ---------*]
                          [*- STA --]
```

where $H(x)$ is the Hilbert transform of the data and $Mod(H(x))$ is the 'envelope' of the seismic signal. *Note that because the Hilbert transform involves performing an FFT of the data it can take significantly longer than the "classic" algorithms for longer seismic signals (>500K pts).*

Value

A vector of values is returned of the same length as the data in the `Trace`.

Note

The returned vector will contain `NA` near the edges of the trace where insufficient data are available to fill the windows. Additional `NA` values will appear for every index that is *skipped over* when the `increment` parameter is greater than one.

For higher resolution channels, picking an increment of `2/sampling_rate` can greatly speed up processing times and still generate reasonable results.

Author(s)

Jonathan Callahan `<jonathan@mazamascience.com>`

References

First break picking (Wikipedia)

Automatic time-picking of first arrivals on large seismic datasets

Automatic first-breaks picking: New strategies and algorithms (Sabbione and Velis 2010)

Adaptive microseismic event detection and automatic time picking (Akram and Eaton 2012)

"Characterization of Global Seismograms Using an Automatic-Picking Algorithm" Bulletin of the Seismological Society of America, Vol. 84, No. 2, pp. 366-376, April 1994 (Earle and Shearer)

See Also

`triggerOnset`

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2010-02-27",tz="GMT")
endtime <- as.POSIXct("2010-02-28",tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]
picker <- STALTA(tr,3,30)

# Plot the trace and overlay the picker
plot(tr)
par(new=TRUE)
plot(picker, type='l', col='red', axes=FALSE, xlab="", ylab="")
mtext("Picker", side=1, line=-8, adj=0.05, col='red')
par(new=FALSE)

## End(Not run)
```

Stream-class *Class* "Stream"

Description

A Stream object containing a list of Trace objects.

Objects from the Class

Objects are typically created by calls to getDataselect.

Slots

url: Object of class "character": URL request used to generate this Stream.

requestedStarttime: Object of class "POSIXct": starttime used when requesting data with getDataselect.

requestedEndtime: Object of class "POSIXct": endtime used when requesting data with getDataselect.

act_flags: Object of class "integer": Accumulators for the act_flags bits in each miniSEED record.

io_flags: Object of class "integer": Accumulators for the io_flags bits in each miniSEED record.

dq_flags: Object of class "integer": Accumulators for the dq_flags bits in each miniSEED record.

timing_qual: Object of class "numeric": Average timing quality associated with miniSEED records.

traces: Object of class "list": List of Trace objects.

Methods

assignQuality `signature(x="Stream", quality="character")`: returns a Stream where the quality code in each trace @id slot is replaced by quality. quality must be one of "R", "D", "Q", "M".

getGaps `signature(x="Stream")`: returns information on data dropouts between `Traces`; see getGaps

getUpDownTimes `signature(x="Stream", min_signal="numeric", min_gap="numeric")`: returns a vector of datetimes associated with channel up/down transitions; see getUpDownTimes

length `signature(x="Stream")`: returns the total number of data points in all `Traces`

max `signature(x="Stream")`: returns the overall data maximum for all data in all `Traces`

median `signature(x="Stream", na.rm= "logical")`: returns the overall data median for all data in all `Traces`

mean `signature(x="Stream")`: returns the overall data mean for all data in all `Traces`

mergeTraces `signature(x="Stream", fillMethod="fillNA")`: returns a new `Stream` object where all `Traces` have been merged into a single `Trace` mergeTraces

min `signature(x="Stream")`: returns the overall data minimum for all data in all `Traces`

multiplyBy `signature(x="Stream", y="numeric")`: returns a new `Stream` object where the data in every `Trace` have been multiplied by y; see multiplyBy

parallelLength `signature(x="Stream")`: returns a vector of data lengths, one for each `Trace`

parallelMax `signature(x="Stream")`: returns a vector of data maxima, one for each `Trace`

parallelMedian `signature(x="Stream", na.rm= "logical")`: returns a vector of data medians, one for each `Trace`

parallelMean `signature(x="Stream")`: returns a vector of data means, one for each `Trace`

parallelMin `signature(x="Stream")`: returns a vector of data minima, one for each `Trace`

parallelRms `signature(x="Stream")`: returns a vector of RMS calculations, one for each `Trace`; see rmsVariance

parallelRmsVariance `signature(x="Stream")`: returns a vector of RMS variance calculations, one for each `Trace`; see rmsVariance

parallelSd `signature(x="Stream", na.rm="logical")`: returns a vector of standard deviation calculations, one for each `Trace`

plot `signature(x="Stream")`: default plot of the merged `Traces` in a `Stream` with appropriate labeling

plotUpDownTimes `signature(x="Stream", min_signal="numeric", min_gap="numeric")`: plots the times at which a `Stream` transitions from data collection to non-collection (on/off); see getUpDownTimes

rms `signature(x="Stream")`: returns the overall Root Mean Square amplitude for all data in all `Traces`; see rmsVariance

rmsVariance `signature(x="Stream")`: returns the overall RMS variance for all data in all `Traces`; see rmsVariance

sd `signature(x="Stream", na.rm="logical")`: returns the overall standard deviations for all data in all `Traces`

slice `signature(x="Stream", starttime="POSIXct", endtime="POSIXct")`: returns a new `Stream` sliced out of an existing `Stream` (see slice)

uniqueIds `signature(x="Stream")`: returns a vector of SNCLQ identifiers, one for each `Trace`

Note

The Stream object is inspired by the Stream class found in the python ObsPy package (https://docs.obspy.org/packages/autogen/obspy.core.stream.Stream.html).

The miniSEED flags and timing_qual values are described in the SEED manual (doi:10.7914/en3h-2318). The "accumulators" contain counts of the number of times each bit flag was set during the parsing of a miniSEED file. These attributes are retained primarily for assessing data quality issues within the EarthScope.

The following code documentation describes how each of the flags is used within miniSEED files:

```
# act_flags
# [1] Calibration signals present
# [2] Time correction applied
# [3] Beginning of an event, station trigger
# [4] End of an event, station detrigger
# [5] A positive leap second happened in this record
# [6] A negative leap second happened in this record
# [7] Event in progress
# [8] Undefined bit set

# io_flags
# [1] Station volume parity error possibly present
# [2] Long record read (possibly no problem)
# [3] Short record read (record padded)
# [4] Start of time series
# [5] End of time series
# [6] Clock locked
# [7] Undefined bit set
# [8] Undefined bit set

# dq_flags
# [1] Amplifier saturation detected
# [2] Digitizer clipping detected
# [3] Spikes detected
# [4] Glitches detected
# [5] Missing/padded data present
# [6] Telemetry synchronization error
# [7] A digital filter may be charging
# [8] Time tag is questionable
```

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")
```

```
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)
min(st)
median(st)
mean(st)
max(st)
sd(st)
rms(st)
rmsVariance(st)

## End(Not run)
```

---

surfaceDistance *Earth surface distance between two points*

Description

The surfaceDistance() function calculates the distance in kilometers between any two lat-lon pairs using the Haversine equation.

Usage

```
surfaceDistance(lat1_deg, lon1_deg, lat2_deg, lon2_deg)
```

Arguments

lat1_deg	latitude 1 (degrees)
lon1_deg	longitude 1 (degrees)
lat2_deg	latitude 2 (degrees)
lon2_deg	longitude 2 (degrees)

Value

Distance in kilometers

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

References

https://en.wikipedia.org/wiki/Haversine_formula

taupTraveltime *Return seismic travel time information using* The TauP Toolkit

Description

The taupTraveltime function returns seismic travel time data computed using The TauP Toolkit (Crotwell, et al., 1999).

software: The TauP Toolkit, version 3.1.0, released 2025-08-15 author: Crotwell, H. Philip <crotwell@seis.sc.edu> repository-code: https://github.com/crotwell/TauP url: http://www.seis.sc.edu/TauP/ repository: doi:10.5281/zenodo.10794857

Usage

```
taupTraveltime(latitude, longitude, depth = 0, staLatitude, staLongitude)
```

Arguments

latitude	latitude of seismic event
longitude	longitude of seismic event
depth	depth of seismic event
staLatitude	latitude of seismic station
staLongitude	longitude of seismic station

Details

taupTraveltime is a wrapper for the travel time functionality of The TauP Toolkit. It returns seismic travel times, calculated using a 1-D spherical model, in a dataframe. It is written to replace the deprecated getTraveltime function that retrieved seismic travel time information from EarthScope web services.

software: The TauP Toolkit, version 3.1.0, released 2025-08-15 author: Crotwell, H. Philip <crotwell@seis.sc.edu> repository-code: https://github.com/crotwell/TauP url: http://www.seis.sc.edu/TauP/ repository: doi:10.5281/zenodo.10794857

A modified version of the TauP Toolkit can be used by setting an environmental variable TAUP_JAR_DIR that points to a local directory containing the jar files in the lib directory of TauP.

Value

A dataframe with the following columns:

```
distance, depth, phaseName, travelTime, rayParam, takeoff, incident
  puristDistance, puristName
```

Rows are ordered by travel time.

Author(s)

Gillian Sharer <gillian.sharer@earthscope.org>

References

Crotwell, H. P., T. J. Owens, J. Ritsema (1999). The TauP Toolkit: Flexible seismic travel-time and ray-path utilities, Seismological Research Letters **70**, 154–160.

Examples

```
## Not run:

# Two days around the "Nisqually Quake"
iris <- new('IrisClient')
starttime <- as.POSIXct("2001-02-27", tz="GMT")
endtime <- starttime + 3600 * 24 * 2

# Find biggest seismic event over these two days -- it's the "Nisqually"
events <- getEvent(iris, starttime, endtime, minmag=5.0)
bigOneIndex <- which(events$magnitude == max(events$magnitude))
bigOne <- events[bigOneIndex[1],]

# Find US stations that are available within an hour of the event
start <- bigOne$time
end <- start + 3600
availability <- getChannel(iris, "US", "", "", "BHZ",
                           starttime=start, endtime=end,
                           latitude=bigOne$latitude, longitude=bigOne$longitude,
                           minradius=0, maxradius=10)

# Get the station the furthest East
minLonIndex <- which(availability$longitude == max(availability$longitude))
snclE <- availability[minLonIndex,]

# Plot the BHZ signal from this station
st <- getDataselect(iris,snclE$network,snclE$station,snclE$location,snclE$channel,
                    start,end)

# Check that there is only a single trace and then plot it
length(st@traces)
tr <- st@traces[[1]]
plot(tr, subsampling=1) # need subsmpling=1 to add vertical lines with abline()

# Find travel times to this station
traveltimes <- taupTraveltime(bigOne$latitude, bigOne$longitude, bigOne$depth,
                              snclE$latitude, snclE$longitude)

# Look at the list
traveltimes

# mark the P and S arrival times
pArrival <- start + traveltimes$travelTime[traveltimes$phaseName=="P"]
```

```
sArrival <- start + traveltimes$travelTime[traveltimes$phaseName=="S"]
abline(v=pArrival, col='red')
abline(v=sArrival, col='blue')

## End(Not run)
```

Trace-class *Class* "Trace"

Description

A Trace object containing a seismic trace – a continuous timeseries.

Objects from the Class

Objects occupy the traces slot of a Stream-class object and are typically populated by calls to getDataselect.

Slots

id: Object of class "character": Unique "SNCL" identifier specifying the Network, Station, Location, Channel and Quality factor associated with this trace: eg. AK.PIN..VEA.M. The id is generated automatically when the trace is first created and is intended for read only.

Sensor: Object of class "character": Instrument name.

InstrumentSensitivity: Object of class "numeric": The total sensitivity for a channel, representing the complete acquisition system expressed as a scalar. Equivalent to SEED stage 0 gain.

SensitivityFrequency: Object of class "numeric": The frequency at which the total sensitivity is correct.

InputUnits: Object of class "character": The units of the data as input from the perspective of data acquisition. After correcting data for this response, these would be the resulting units.

stats: Object of class "TraceHeader": Container with metadata information describing the trace. (see TraceHeader-class)

data: Object of class "numeric": Vector of data values.

Methods

assignQuality signature(x="Trace", quality="character"): returns a Trace where the quality code in the trace @id slot is replaced by quality. quality must be one of "R", "D", "Q", "M".

as.vector signature(x="Trace"): returns the data slot; equivalent to x@data

DDT signature(x="Trace", demean="logical", detrend="logical", taper="numeric"): returns a new trace that has been 'cleaned up' for further processing by applying demean, detrend, and taper techniques (see DDT)

envelope signature(x="Trace"): returns the envelope of the seismic signal (see envelope)

isDC `signature(x="Trace")`: returns `TRUE` if trace data consist of a DC signal

length `signature(x="Trace")`: returns the length of the data; equivalent to `length(x@data)`

max `signature(x="Trace")`: returns the maximum value of the data; equivalent to `max(x@data)`

median `signature(x="Trace", na.rm="logical")`: returns the median value of the data; equivalent to `median(x@data)`

mean `signature(x="Trace")`: returns the mean value of the data; equivalent to `mean(x@data)`

min `signature(x="Trace")`: returns the minimum value of the data; equivalent to `min(x@data)`

multiplyBy `signature(x="Trace", y="numeric")`: returns a new `Trace` where the data have been multiplied by y (see multiplyBy)

plot `signature(x="Trace")`: default plot of the `Trace` data with appropriate labeling

rms `signature(x="Trace")`: returns the Root Mean Square amplitude of the data (see rms)

rmsVariance `signature(x="Trace")`: returns the RMS variance of the data (see rmsVariance)

sd `signature(x="Trace", na.rm="logical")`: returns the standard deviation of the data; equivalent to `sd(x@data)`

slice `signature(x="Trace", starttime="POSIXct", endtime="POSIXct")`: returns a new `Trace` subset of an existing `Trace` (see slice)

STALTA `signature(x="Trace",staSecs="numeric",ltaSecs="numeric",algorithm="character", ...)`: returns the STALTA picker result (see STALTA)

triggerOnset `signature(x="Trace", picker="numeric", threshold="numeric", ...)`: returns the time or index of an event onset as determined by the STALTA picker (see triggerOnset)

Note

The `Trace` object is inspired by the `Trace` class found in the python ObsPy package (https://docs.obspy.org/packages/autogen/obspy.core.trace.Trace.html).

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

# Set the starttime and endtime
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Get the first trace and generate some statistics
tr1 <- st@traces[[1]]
min(tr1)
```

```
median(tr1)
mean(tr1)
max(tr1)
sd(tr1)
rms(tr1)
rmsVariance(tr1)

## End(Not run)
```

TraceHeader-class *Class* "TraceHeader"

Description

A container for metadata associated with a Trace object. Originally populated by information in the miniseed trace header; it now has the option of including additional station and channel metadata.

Objects from the Class

Objects can be created by calls of the form new("TraceHeader", headerList, headerLine, ...). The stats slot of a Trace object will contain a TraceHeader object, typically populated by a webservice request. (see IrisClient-class)

Slots

sampling_rate: Object of class "numeric": Sampling rate in hertz.

delta: Object of class "numeric": Sample interval in seconds.

calib: Object of class "numeric": Calibration factor.

npts: Object of class "integer": Number of sample points.

network: Object of class "character": Seismic network name.

location: Object of class "character": Location code.

station: Object of class "character": Station name.

channel: Object of class "character": Channel code.

quality: Object of class "character": Data quality code.

starttime: Object of class "POSIXct": Start time.

endtime: Object of class "POSIXct": End time.

latitude: Object of class "numeric": Latitude.

longitude: Object of class "numeric": Longitude.

elevation: Object of class "numeric": Elevation.

depth: Object of class "numeric": Depth.

azimuth: Object of class "numeric": Azimuth.

dip: Object of class "numeric": Dip.

processing: Object of class "list": Information strings describing processing applied to this trace.

Methods

as.headerLine `signature(obj = "TraceHeader")`: Prints out the information in the `TraceHeader` as an ascii header line, not including any station and channel metadata not found in the miniseed trace header, e.g.,

```
TIMESERIES LD_POTS__HHZ_M, 351 samples, 100.503 sps, \
2012-01-29T00:00:00.006000, SLIST, INTEGER, COUNTS
```

show `signature(object = "TraceHeader")`: Prettyprints the information in the `TraceHeader`

Note

The `TraceHeader` object is inspired by the `Stats` class found in the python ObsPy package (https://docs.obspy.org/packages/autogen/obspy.core.trace.Stats.html).

Retaining the ObsPy class name `Stats` would have generated a tremendous amount of confusion in the context of R. Instead, the name `TraceHeader` has been adopted. Nevertheless, the `TraceHeader` object still lives in the `Trace@stats` slot to retain as much similarity to ObsPy as possible.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")

# Get the waveform
st <- getDataselect(iris,"AK","PIN","","BHZ",starttime,endtime)

# Get the first trace and show the associated metadata
tr1 <- st@traces[[1]]
show(tr1@stats)

## End(Not run)
```

`transferFunctionSpectra` *function to retrieve response values to use for input to the IRISMustangMetrics::transferFunctionMetric*

Description

The transferFunctionSpectra function returns a frequency-amplitude-phase response from the service.earthscope.org/iris/ws/evalresp web service for a seismic Stream object using sampling_rate to determine frequency limits. The IRISMustangMetrics::transferFunctionMetric expects this output as evalresp input.

Usage

```
transferFunctionSpectra(st,sampling_rate)
```

Arguments

st	a Stream object
sampling_rate	sample rate

Details

The transferFunctionSpectra /determines the minfreq, maxfreq, and nfreq for input to the getEvalresp function based on input sample rate. Other getEvalresp inputs are units='def' (default) and output='fap' (frequency-amplitude-phase).

transferFunctionSpectra will always call getEvalresp using the service.earthscope.org/irisws/evalresp web service. The IRISMustangMetrics::transferFunctionMetric expects this output as evalresp input.

Value

Output is a dataframe with columns named:

```
freq, amp, phase
```

Author(s)

Mary Templeton <mary.templeton@earthscope.org>

See Also

getEvalresp,

Examples

```
## Not run:
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# Get seismic data
starttime <- as.POSIXct("2011-05-05", tz="GMT")
endtime <- starttime + 1*24*3600
st <- getDataselect(iris,"IU","GRFO","--","BHE",starttime,endtime)
sampling_rate <- 20
```

```
# Generate power spectral density for each hour long segment
evalresp  <- transferFunctionSpectra(st,sampling_rate)


## End(Not run)
```

`triggerOnset` *Event onset triggering*

Description

The `triggerOnset` method of `Trace` objects uses the numeric vector returned by the `STALTA` "first break picking" method and a user selected threshold to determine the arrival time of a seismic event.

Usage

```
triggerOnset(x, picker, threshold, index)
```

Arguments

`x`	a `Trace` object
`picker`	results from applying the `STALTA` method to this trace
`threshold`	optional numeric value of the threshold at which triggering should occur
`index`	optional logical to return the index (rather than the time) of event onset (default=`FALSE`)

Details

This method simply identifies the point at which the `picker` first rises above the `threshold`.

When no `threshold` is supplied, an appropriate value is calculated from the picker with:

`threshold <- quantile(picker,0.999,na.rm=TRUE)`.

Value

A single value is returned identifying the onset of the seismic event or `NA` if none is detected. The returned value wil be a `POSIXct` time by defult or a numeric index if `index=TRUE`.

Note

The appropriate value for the threshold will depend upon the exact STA/LTA algorithm used and the noise level in the signal.

Author(s)

Jonathan Callahan <jonathan@mazamascience.com>

See Also

STALTA

Examples

```
## Not run:
# Open a connection to EarthScope webservices
iris <- new("IrisClient")

starttime <- as.POSIXct("2010-02-27 06:00:00",tz="GMT")
endtime <- as.POSIXct("2010-02-27 09:00:00",tz="GMT")

# Get the waveform
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]
picker <- STALTA(tr,3,30)

# Identify the onset of the event
to <- triggerOnset(tr,picker)

plot(tr)
abline(v=to, col='red', lwd=2)

## End(Not run)
```

---

unHistogram *Histogram to Vector*

Description

If vec represents a set of binned counts of incrementing values (ascending) return a vector of associated bin values with the proper count of each value. Intended for internal use.

Usage

```
unHistogram(vec, startVal, incr)
```

Arguments

vec	a histogram vector or ordered set of binned counts
startVal	the initial value of the first bin element
incr	the increment rate of each subsequent bin value

Value

A vector of bin values with appropriate counts of each.

Author(s)

Rob Casey <rob.casey@earthscope.org>

Index

∗ **classes**
IrisClient-class, 56
Stream-class, 84
Trace-class, 90
TraceHeader-class, 92
∗ **methods**
assignQuality, 12
basicStats, 12
butterworth, 14
DDT, 18
envelope, 20
eventWindow, 21
getGaps, 37
getUpDownTimes, 52
hilbert, 53
hilbertFFT, 55
mergeTraces, 61
multiplyBy, 65
rms, 77
slice, 80
STALTA, 81
triggerOnset, 95
unHistogram, 96
∗ **spectra**
crossSpectrum, 16
McNamaraBins, 58
McNamaraPSD, 59
noiseMatrix2PdfMatrix, 66
noiseModels, 67
psdDF2NoiseMatrix, 68
psdList, 69
psdList2NoiseMatrix, 71
psdPlot, 73
psdStatistics, 74
transferFunctionSpectra, 93
∗ **webservices**
getAvailability, 23
getChannel, 25
getDataAvailability, 28
getDataselect, 30
getDistaz, 32
getEvalresp, 34
getEvent, 35
getNetwork, 38
getRotation, 41
getSNCL, 42
getStation, 43
getTimeseries, 46
getTraveltime, 48
getUnavailability, 50

as.headerLine *(*TraceHeader-class*)*, 92
as.headerLine,TraceHeader-method *(*TraceHeader-class*)*, 92
as.vector,Trace-method *(*Trace-class*)*, 90
assignQuality, 12
assignQuality,Stream,character-method *(*assignQuality*)*, 12
assignQuality,Trace,character-method *(*assignQuality*)*, 12

basicStats, 12
butterworth, 14
butterworth,Trace,numeric,missing,numeric,missing-method *(*butterworth*)*, 14
butterworth,Trace,numeric,numeric,missing,missing-method *(*butterworth*)*, 14
butterworth,Trace,numeric,numeric,numeric,character-method *(*butterworth*)*, 14
butterworth,Trace,numeric,numeric,numeric,missing-method *(*butterworth*)*, 14

crossSpectrum, 16

DDT, 18, *20*, *54*, *90*
DDT,Trace,logical,logical,numeric-method *(*DDT*)*, 18
DDT,Trace,missing,missing,missing-method *(*DDT*)*, 18

envelope, 20, *90*
envelope,Trace-method (envelope), 20
eventWindow, 21
eventWindow,Trace,numeric,missing,missing-method (eventWindow), 21
eventWindow,Trace,numeric,missing,numeric-method (eventWindow), 21
eventWindow,Trace,numeric,numeric,missing-method (eventWindow), 21
eventWindow,Trace,numeric,numeric,numeric-method (eventWindow), 21

getAvailability, 23, *27*, *51*, *57*
getAvailability,IrisClient,character,character,character,character,POSIXct,POSIXct,logical-method (getAvailability), 23
getAvailability,IrisClient,character,character,character,character,POSIXct,POSIXct,missing-method (getAvailability), 23
getChannel, *24*, 25, *51*, *57*
getChannel,IrisClient,character,characterORmissing,characterORmissing,characterORmissing,POSIXctORmissi (getChannel), 25
getDataAvailability, 28, *57*
getDataAvailability,IrisClient,character,character,character,character,POSIXct,POSIXct-method (getDataAvailability), 28
getDataselect, 30, *37*, *43*, *47*, *52*, *57*, *64*, *84*, *90*
getDataselect,IrisClient,character,character,character,character,POSIXct,POSIXct-method (getDataselect), 30
getDistaz, 32, *57*
getDistaz,IrisClient,numeric,numeric,numeric,numeric-method (getDistaz), 32
getEvalresp, 34, *57*, *60*, *68*, *71*, *94*
getEvalresp,IrisClient,character,character,character,character,POSIXct-method (getEvalresp), 34
getEvent, 35, *57*
getEvent,IrisClient,POSIXctORmissing,POSIXctORmissing-method (getEvent), 35
getGaps, 37, *85*
getGaps,Stream,missing-method (getGaps), 37
getGaps,Stream,numeric-method (getGaps), 37
getNetwork, 38, *57*
getNetwork,IrisClient,characterORmissing,characterORmissing,characterORmissing,characterORmissing,POSIX (getNetwork), 38
getRotation, 41
getRotation,IrisClient,character,character,character,character,character,POSIXct,POSIXct,character-method (getRotation), 41
getSNCL, *32*, 42, *47*, *57*
getSNCL,IrisClient,character,POSIXct,POSIXct-method (getSNCL), 42
getStation, 43, *57*
getStation,IrisClient,character,characterORmissing,charact (getStation), 43
getTimeseries, 46
getTimeseries,IrisClient,character,character,character,cha (getTimeseries), 46
getTraveltime, 48, *57*
getTraveltime,IrisClient,numeric,numeric,numeric,numeric,n (getTraveltime), 48
getUnavailability, *24*, *27*, 50, *57*
getUnavailability,IrisClient,character,character,character (getUnavailability), 50
getUnavailability,IrisClient,character,character,character (getUnavailability), 50
getUpDownTimes, 52, *63*, *85*
getUpDownTimes,Stream,missing,missing-method (getUpDownTimes), 52
getUpDownTimes,Stream,numeric,numeric-method (getUpDownTimes), 52

hilbert, 53
hilbert,Trace-method (hilbert), 53
hilbertFFT, 55

initialize,IrisClient-method (IrisClient-class), 56
initialize,Trace-method (Trace-class), 90
initialize,TraceHeader-method (TraceHeader-class), 92
IrisClient-class, 56, *92*
IRISSeismic (IRISSeismic-package), 3
IRISSeismic-deprecated, 58
IRISSeismic-package, 3
isDC (Trace-class), 90
isDC,Trace-method (Trace-class), 90

length (basicStats), 12
length,Stream-method (basicStats), 12
length,Trace-method (basicStats), 12

max (basicStats), 12
max,Stream-method (basicStats), 12
max,Trace-method (basicStats), 12
McNamaraBinaraster, 58, *61*
McNamaraPSD, *17*, *59*, 59, *67*, *69*, *70*, *72*, *73*, *75*
mean (basicStats), 12

mean,Stream-method (basicStats), 12
mean,Trace-method (basicStats), 12
median (basicStats), 12
median,Stream,logical-method (basicStats), 12
median,Stream,missing-method (basicStats), 12
median,Trace,logical-method (basicStats), 12
median,Trace,missing-method (basicStats), 12
mergeTraces, 61, *85*
mergeTraces,Stream,character-method (mergeTraces), 61
mergeTraces,Stream,missing-method (mergeTraces), 61
mergeUpDownTimes, 62
mergeUpDownTimes,NULL,POSIXct,logical-method (mergeUpDownTimes), 62
mergeUpDownTimes,NULL,POSIXct,missing-method (mergeUpDownTimes), 62
mergeUpDownTimes,POSIXct,NULL,logical-method (mergeUpDownTimes), 62
mergeUpDownTimes,POSIXct,NULL,missing-method (mergeUpDownTimes), 62
mergeUpDownTimes,POSIXct,POSIXct,logical-method (mergeUpDownTimes), 62
mergeUpDownTimes,POSIXct,POSIXct,missing-method (mergeUpDownTimes), 62
min (basicStats), 12
min,Stream-method (basicStats), 12
min,Trace-method (basicStats), 12
miniseed2Stream, 64, *77*
multiplyBy, 65, *85*, *91*
multiplyBy,Stream,numeric-method (multiplyBy), 65
multiplyBy,Trace,numeric-method (multiplyBy), 65

noiseMatrix2PdfMatrix, 66
noiseModels, 67

parallelLength (basicStats), 12
parallelLength,Stream-method (basicStats), 12
parallelMax (basicStats), 12
parallelMax,Stream,logical-method (basicStats), 12
parallelMax,Stream,missing-method (basicStats), 12
parallelMean (basicStats), 12
parallelMean,Stream,logical-method (basicStats), 12
parallelMean,Stream,missing-method (basicStats), 12
parallelMedian (basicStats), 12
parallelMedian,Stream,logical-method (basicStats), 12
parallelMedian,Stream,missing-method (basicStats), 12
parallelMin (basicStats), 12
parallelMin,Stream,logical-method (basicStats), 12
parallelMin,Stream,missing-method (basicStats), 12
parallelRms (rms), 77
parallelRmsVariance (rms), 77
parallelRmsVariance,Stream,logical-method (basicStats), 12
parallelRmsVariance,Stream,missing-method (basicStats), 12
parallelSd (basicStats), 12
parallelSd,Stream,logical-method (basicStats), 12
parallelSd,Stream,missing-method (basicStats), 12
plot,Stream-method (Stream-class), 84
plot,Trace-method (Trace-class), 90
plotUpDownTimes, *53*, *63*
plotUpDownTimes (Stream-class), 84
plotUpDownTimes,POSIXct,missing,missing-method (Stream-class), 84
plotUpDownTimes,POSIXct,missing,numeric-method (Stream-class), 84
plotUpDownTimes,POSIXct,numeric,missing-method (Stream-class), 84
plotUpDownTimes,POSIXct,numeric,numeric-method (Stream-class), 84
plotUpDownTimes,Stream,missing,missing-method (Stream-class), 84
plotUpDownTimes,Stream,missing,numeric-method (Stream-class), 84
plotUpDownTimes,Stream,numeric,missing-method (Stream-class), 84
plotUpDownTimes,Stream,numeric,numeric-method (Stream-class), 84

psdDF2NoiseMatrix, *67*, 68, *72*
psdList, *60*, *61*, *67*, *69*, 69, *72*, *73*, *75*
psdList2NoiseMatrix, *68–70*, 71
psdPlot, *67*, *69*, *70*, *72*, 73, *75*
psdStatistics, *67–70*, *72*, *73*, 74

readMiniseedFile, *65*, 76
rms, 77, *91*
rms,Stream,logical-method *(*basicStats*)*, 12
rms,Stream,missing-method *(*basicStats*)*, 12
rms,Trace,logical-method *(*rms*)*, 77
rms,Trace,missing-method *(*rms*)*, 77
rmsVariance, *85*, *91*
rmsVariance *(*rms*)*, 77
rmsVariance,Stream,logical-method *(*basicStats*)*, 12
rmsVariance,Stream,missing-method *(*basicStats*)*, 12
rmsVariance,Trace,logical-method *(*rms*)*, 77
rmsVariance,Trace,missing-method *(*rms*)*, 77
rotate2D, 79

sd *(*basicStats*)*, 12
sd,Stream,logical-method *(*basicStats*)*, 12
sd,Stream,missing-method *(*basicStats*)*, 12
sd,Trace,logical-method *(*basicStats*)*, 12
sd,Trace,missing-method *(*basicStats*)*, 12
show,TraceHeader-method *(*TraceHeader-class*)*, 92
slice, 80, *85*, *91*
slice,Stream,POSIXct,POSIXct-method *(*Stream-class*)*, 84
slice,Trace,POSIXct,POSIXct-method *(*slice*)*, 80
STALTA, *22*, 81, *91*, *96*
STALTA,Trace,missing,missing,missing,missing,missing,missing,missing-method *(*STALTA*)*, 81
STALTA,Trace,numeric,numeric,character,logical,logical,numeric,numeric-method *(*STALTA*)*, 81
STALTA,Trace,numeric,numeric,character,missing,missing,missing,missing-method *(*STALTA*)*, 81
STALTA,Trace,numeric,numeric,missing,missing,missing,missing,missing-method *(*STALTA*)*, 81
Stream *(*Stream-class*)*, 84
Stream-class, 84, *90*
surfaceDistance, 87

taupTraveltime, 88
Trace, *64*, *76*
Trace *(*Trace-class*)*, 90
Trace-class, 90
TraceHeader *(*TraceHeader-class*)*, 92
TraceHeader-class, *90*, 92
transferFunctionSpectra, 93
triggerOnset, *22*, *83*, *91*, 95
triggerOnset,Trace,numeric-method *(*triggerOnset*)*, 95

unHistogram, 96
uniqueIds *(*Stream-class*)*, 84
uniqueIds,Stream-method *(*Stream-class*)*, 84