Package 'IRISMustangMetrics'

September 16, 2015

Type Package

Version 1.0.1

Title Statistics and Metrics for Seismic Data

Author Jonathan Callahan [aut, cre],
Rob Casey [aut],
Mary Templeton [aut]

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

Depends R (>= 3.1.0), IRISSeismic (>= 1.0.5)

Imports methods, pracma, RCurl, seismicRoll (>= 1.0.1), signal, stringr, XML

Collate Class-Metric.R BSSUtils.R basicStatsMetric.R correlationMetric.R crossCorrelationMetric.R dailyDCOffsetMetric.R DCOffsetTimesMetric.R gapsMetric.R PSDMetric.R SNRMetric.R spikesMetric.R STALTAMetric.R stateOfHealthMetric.R upDownTimesMetric.R transferFunctionMetric.R

Description Classes and functions for metrics calculation as part of the 'IRIS DMC MUSTANG' project. The functionality in this package builds upon the base classes of the 'IrisSeismic' package. Metrics include basic statistics as well as higher level 'health' metrics that can help identify problematic seismometers.

License GPL (>= 2)

Repository CRAN

NeedsCompilation no

Date/Publication 2015-09-16 10:04:57

R topics documented:

`seismicMetrics-package`

Utilities for calculating seismic metrics from IRIS DMC data

Description

This package provides S4 classes and functions for calculating metrics from seismological data available from the IRIS Data Management Center (DMC) (http://www.iris.edu/dms/nodes/dmc/). This package is part of the MUSTANG project and is intended **for DMC internal use only**.

Introduction

The **seismicMetrics** package depends upon the **seismic** package which defines new S4 classes and methods for manipulating seismic data. Please see the "seismic-intro" vignette for introductory examples on using **seismic**.

History

version 1.0.0 – First Public Release

Author(s)

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

References

IRIS DMC web services: http://service.iris.edu/

Examples

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

# Get the seismic data
starttime <- as.POSIXct("2010-02-27 06:45:00",tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:45:00",tz="GMT")
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)

# Apply a metric and show the results
metricList <- basicStatsMetric(st)
dummy <- lapply(metricList, show)
```

---

basicStatsMetric *Min, median, mean, and rms max of a signal*

Description

The basicStatsMetric() function calculates the `min, median, mean, max` and `rms` for the incoming seismic signal.

Usage

```
basicStatsMetric(st)
```

Arguments

`st`	a `Stream` object containing a seismic signal

Details

This metric merely applies the `min, median, mean` and `max` methods of `Stream` objects to the `st` parameter to calculate the following metrics:

- `sample_min`
- `sample_median`
- `sample_mean`
- `sample_max`
- `sample_rms`

Any error messages generated in the process will pass through untrapped.

Value

A list of `SingleValueMetric` objects is returned.

Note

See the `seismic` package for documentation on `Stream` objects and the `getDataselect` method.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24 00:00:00", tz="GMT")
endtime <- as.POSIXct("2012-01-25 00:00:00", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Calculate some metrics and show the results
metricList <- basicStatsMetric(st)
dummy <- lapply(metricList, show)
```

---

convertBssErrors *Generate Human Readable MUSTANG Errors*

Description

The MUSTANG database is in charge of storing the results of metrics calculations and is accessed through a webservice API. The`convertBssErrors` function extracts pertinent error information from the HTML returned by MUSTANG on error conditions.

Usage

```
convertBssErrors(err_msg)
```

Arguments

`err_msg`	error text received from the MUSTANG

Value

Aext string with the `root cause` extracted from the MUSTANG HTML Java error dump.

Author(s)

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

See Also

SingleValueMetric-class, metricList2Xml, getMetricsXml, getBssMetricList,

---

`correlationMetric` *Correlation between channels*

Description

The correlationMetric() function calculates the correlation between two streams of seismic data.

Usage

```
correlationMetric(st1, st2)
```

Arguments

st1	a Stream object containing a seismic signal
st2	a Stream object containing a seismic signal

Details

The correlation returned is a value in the range [0-1]. This 'pearson r' correlation is a measure of the strength and direction of the linear relationship between two variables that is defined as the (sample) covariance of the variables divided by the product of their (sample) standard deviations.

Missing values are handled by casewise deletion with the following R code:

```
cor(x,y,use="na.or.complete")
```

Value

A list with a single SingleValueMetric object is returned. The metric name is cross_talk.

Note

Seismic streams passed to correlationMetric must have the same network and station, must cover the same time range and must have the same sampling rate.

The metricList generated for this two-channel metric will have a SNCL code of the form: N.S.L1:L2.C1:C2.Q.

Author(s)

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

Examples

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

# Get seismic traces
starttime <- as.POSIXct("2013-03-01", tz="GMT")
endtime <- as.POSIXct("2013-03-02",tz="GMT")
stZ <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
st1 <- getDataselect(iris,"IU","ANMO","00","BH1",starttime,endtime)
st2 <- getDataselect(iris,"IU","ANMO","00","BH2",starttime,endtime)

# Calculate correlationMetric
correlationMetric(stZ,st1)[[1]]
correlationMetric(stZ,st2)[[1]]
correlationMetric(st1,st2)[[1]]

## End(Not run)
```

`createBssUrl` *Create URL to retrieve measurements from the MUSTANG BSS*

Description

The `createBssUrl` method of the `IrisClient` returns a URL that can be used to make a request of the MUSTANG BSS (Backend Storage System).

Usage

```
createBssUrl(obj, network, station, location, channel,
             starttime, endtime, metricName, constraint, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string containing one or more comma separated metric names
`constraint`	a character string containing value constraints
`url`	optional url of the BSS measurements service

Details

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Value

A character string containing a BSS request URL

Author(s)

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

See Also

`getSingleValueMetrics`

Examples

```
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2013-06-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_min,sample_mean"

# Get the measurement dataframe
url <- createBssUrl(iris,"IU","ANMO","00","BHZ",
                    starttime,endtime,metricName)

# This URL can be pasted into a web browser to see the BSS return values
```

`crossCorrelationMetric` *Correlation between channels*

Description

The crossCorrelationMetric() function calculates the `peak_correlation` and `peak_lag` associated with two streams of seismic data.

Usage

```
crossCorrelationMetric(st1, st2, maxLagSecs=10, filter=signal::butter(2,0.2))
```

Arguments

st1	a Stream object containing a seismic signal
st2	a Stream object containing a seismic signal
maxLagSecs	maximum number of seconds of lag to use
filter	a **signal** package filter to be applied before cross-correlating

Details

Details of the algorithm are as follows:

- Both signals are demeaned and detrended
- If one signal has a higher sampling rate, it is decimated to the lower sampling rate using an IIR filter see(signal::decimate).
- Both signals are filtered, by default with a Butterworth low pass filter see(signal::filter).
- Signals are cross-correlated using the ccf() function.

The maximum absolute correlation is saved as peak_correlation while the lag at peak correlation is saved as peak_lag.

Note: For cross-correlation, seismic signals must not have any gaps – they must be contained in a single Trace object.

Value

A list with two SingleValueMetric objects is returned. The metric names are peak_correlation and peak_lag.

Note

The metricList generated for this two-channel metric will have an additional sncl2 attribute identifying the SNCL in st2.

Author(s)

Jonathan Callahan <jonathan.s.callahan@gmail.com> (R code), Mary Templeton <met@iris.washington.edu> (algorithm)

Examples

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

# Get the same signal, shifted by 3 seconds
starttime <- as.POSIXct("2013-11-12 07:09:45",tz="GMT")
endtime <- starttime + 600
st1 <- getSNCL(iris,"NM.SLM.00.BHZ",starttime,endtime)
st2 <- getSNCL(iris,"NM.SLM.00.BHZ",starttime+3,endtime+3)

# Cross-correlate
```

```
crossCorrelationMetric(st1,st2)

#[[1]]
# SingleValueMetric
#  metric:        peak_correlation
#  snclq:         NM.SLM.00.BHZ.M
#  starttime:     2013-11-12 07:09:45
#  endtime:       2013-11-12 07:19:45
#  value:         0.999
#  snclq2:    NM.SLM.00.BHZ.M
#
#[[2]]
#SingleValueMetric
#  metric:        peak_lag
#  snclq:         NM.SLM.00.BHZ.M
#  starttime:     2013-11-12 07:09:45
#  endtime:       2013-11-12 07:19:45
#  value:         3.000
#  snclq2:    NM.SLM.00.BHZ.M
```

`dailyDCOffsetMetric` *DC Offset Detection*

Description

The dailyDCOffsetMetric() function identifies days with a jump in the signal mean.

Usage

```
dailyDCOffsetMetric(df, offsetDays=5,
                    outlierWindow=7,
                    outlierThreshold=6.0,
                    outlierSelectivity=0.1)
```

Arguments

`df`	a dataframe containing `sample_mean` values obtained with getSingleValueMetrics()
`offsetDays`	number of days used in calculating weighting factors
`outlierWindow`	window size passed to findOutliers() function in the **seismic** package
`outlierThreshold`	detection threshold passed to findOutliers() function in the **seismic** package
`outlierSelectivity`	selectivity passed to findOutliers() function in the **seismic** package

Details

This algorithm calculates lagged differences of the daily mean timeseries over a window of `offsetDays` days. Shifts in the mean that are persistent and larger than the typical standard deviation of daily means will generate higher metric values.

Details of the algorithm are as follows

```
# data0 = download 12+ months of daily means (in the 'df' dataframe)
# data1 = remove outliers using MAD outlier detection with the 'outlier' arguments specified
# data2 = replace outliers with rolling median values using a 7 day window
# weights = calculate absolute lagged differences with 1-N day lags (big jumps have large values)
# metric0 = multiply the lagged differences together and take the N'th root
# stddev0 = calculate the rolling standard deviation of data2 with a N-day window
# METRIC = divide metric0 by the median value of stddev0
```

Value

A list is returned with a `MultipleTimeValueMetric` object for each day in the incoming dataframe..

Note

After initial testing on stations in the IU network, a metric value > 10 appears to be indicative of a DC offset shift.

Author(s)

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

See Also

`getSingleValueMetrics`

`DCOffsetTimesMetric` *DC Offset Detection*

Description

The DCOffsetTimesMetric() function returns times where a shift in the signal mean is detected.

Usage

```
DCOffsetTimesMetric(st, windowSecs, incrementSecs, threshold)
```

Arguments

`st`	a `Stream` object containing a seismic signal
`windowSecs`	chunk size (secs) used in DCOffset calculations (default=`1800`)
`incrementSecs`	increment (secs) for starttime of sequential chunks (default=`windowSecs/2`)
`threshold`	threshold used in the detection metric (default=`0.9`)

Details

Conceptually, this algorithm asserts: If the difference in means between sequential chunks of seismic signal is greater than the typical std dev of a chunk then this marks a DC offset shift.

Details of the algorithm are as follows

```
# Merge all traces in the time period, filling gaps with missing values
# Break up the signal into windowSecs chunks spaced incrementSecs apart
# For each chunk calculate:
#     signal mean, signal standard deviation
# Resulting mean and std dev arrays are of length 47 for 24 hours of signal
# Metric = abs(lagged difference of chunk means) / mean(chunk std devs)
# DC offset = times when Metric > threshold
```

Value

A list with a single `MultipleTimeValueMetric` object is returned.

Note

The denominator of this metric was tested with both `mean(chunk std devs)` and with `median(chunk std devs)` to identify a "typical" value for the chunk standard deviation. It was found that using `median` resulted false offset detects whenever there was a large seismic signal in an otherwise lo-noise signal.

Author(s)

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

Examples

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

# Get a signal with a DC offset problem
starttime <- as.POSIXct("2012-10-26",tz="GMT")
endtime <- starttime + 2*24*3600
st <- getDataselect(iris,"IU","TARA","00","BHZ",starttime,endtime)

# Calculate the metric
metricList <- DCOffsetTimesMetric(st)

# Extract values from the first element of the list
offsetTimes <- metricList[[1]]@values

# Plot the signal and mark locations where a DC offset was detected
plot(st)
abline(v=offsetTimes,col='red')

## End(Not run)
```

gapsMetric *Gaps and overlaps in a signal*

Description

The gapsMetric() function calculates metrics associated with gaps and overlaps in a siesmic signal, *i.e.* when st consists of more than one Trace.

Usage

```
gapsMetric(st)
```

Arguments

st	a Stream object containing a seismic signal

Details

This function uses the output of the getGaps method of Stream objects to calculate the following metrics:

num_gaps: number of gaps found in st

max_gap: legnth of maximum gap (sec) found in st

num_overlaps: number of overlaps found in st

max_overlap: legnth of maximum overlap (sec) found in st

percent_availability: percentage of total requested time for which a signal is available

The requestedStarttime and requestedEndtime slots for the Stream are used to determine gaps before the start of the firstor after the end of the last Trace in the Stream.

Value

A list of SingleValueMetric objects is returned.

Note

See the seismic package for documentation on Stream objects and the getDataselect method.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Calculate the gaps metrics and show the results
metricList <- gapsMetric(st)
dummy <- lapply(metricList, show)
```

`getBssMetricList` *Retrieve measurements XML from the MUSTANG BSS and convert them to a metricList*

Description

The `getBssMetricList` method makes a request of the MUSTANG BSS (Backend Storage System) and returns a list of `_Metric` objects.

Usage

```
getBssMetricList(obj, network, station, location, channel,
                 starttime, endtime, metricName, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string identifying the name of the metric stored in the BSS
`url`	optional url of the BSS measurements service

Details

This method calls on getMetricsXml to communicate with the BSS and obtain an XML reponse. This response is then processed and used to create `_Metric` objects which are returned as a metricList.

Error returns from the BSS will stop evaluation and throw an error message.

Value

A list of _Metric objects is returned.

Author(s)

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

See Also

getMetricsXml

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

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

# Get the metricList
metricList <- getBssMetricList(iris,"AK","PIN","","",starttime,endtime,
                               metricName="sample_mean")
show(metricList)

## End(Not run)
```

---

getLatencyValuesXml *Retrieve instantaneous data latency XML from the MUSTANG latency servlet.*

Description

The getLatencyValuesXml method makes a request of the IRIS DMC latency servlet and returns the unprocessed data latency XML for submission to the MUSTANG BSS (Backend Storage System).

Usage

```
getLatencyValuesXml(obj, network, station, location, channel, url)
```

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
url	optional url of the latency web service

Details

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

Value

An XML string is returned.

Note

This method is called by `latencyMetric.Rscript`.

Author(s)

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

See Also

IrisClient-class

---

getMetricsXml *Retrieve measurements XML from the MUSTANG BSS*

Description

The `getMetricsXml` method makes a request of the MUSTANG BSS (Backend Storage System) and returns a character string with the response XML.

Usage

```
getMetricsXml(obj, network, station, location, channel,
                   starttime, endtime, metricName, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string identifying the name of the metric stored in the BSS
`url`	optional url of the BSS measurements service

Details

The default BSS measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

This method returns raw XML which is not that useful by itself. Users should instead use the getBssMetricList method which calls this function and returns a list `_Metric` objects.

Error returns from the BSS will stop evaluation and throw an error message.

Value

A character string with the XML response from the BSS is returned.

Author(s)

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

See Also

getBssMetricList

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

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

# Get the measurement XML
xml <- getMetricsXml(iris,"AK","PIN","","VEA",
                          starttime,endtime,metricName="sample_mean",
                          url="http://service.iris.edu/mustang/measurements/1/query?")

## End(Not run)
```

`getPsdMetrics` *Retrieve measurements from the MUSTANG BSS*

Description

The `getPsdMetrics` method of the `IrisClient` makes a request of the MUSTANG BSS (Backend Storage System) and returns a dataframe containing Power Spectral Density (PSD) measurements.

Usage

```
getPsdMetrics(obj, network, station, location, channel,
                   starttime, endtime, url)
```

Arguments

obj	an IrisClient object
network	a character string with the two letter seismic network code
station	a character string with the station code
location	a character string with the location code
channel	a character string with the three letter channel code
starttime	a POSIXct class specifying the starttime (GMT)
endtime	a POSIXct class specifying the endtime (GMT)
url	optional url of the BSS measurements service

Details

The default BSS measurement service when url is not specified is:

http://service.iris.edu/mustang/measurements/1/query?

The metric name is automatically specified by appending metric=psd to this url.

Data returned from the BSS are converted into an R dataframe.

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

Value

A dataframe with the following columns:

target, starttime, endtime, frequency, amplitude, phase

Author(s)

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

See Also

getBssMetricList

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

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

# Get the measurement XML
xml <- getPsdMetrics(iris,"AK","PIN","","VEA",
                          starttime,endtime,metricName="sample_mean",
                          url="http://service.iris.edu/mustang/measurements/1/query?")

## End(Not run)
```

`getSingleValueMetrics` *Retrieve measurements from the MUSTANG BSS*

Description

The `getSingleValueMetrics` method of the `IrisClient` makes a request of the MUSTANG database and returns a dataframe containing metrics that are stored as single values, e.g. sample_max, sample_min, etc..

Usage

```
getSingleValueMetrics(obj, network, station, location, channel,
                          starttime, endtime, metricName, constraint, url)
```

Arguments

`obj`	an `IrisClient` object
`network`	a character string with the two letter seismic network code
`station`	a character string with the station code
`location`	a character string with the location code
`channel`	a character string with the three letter channel code
`starttime`	a POSIXct class specifying the starttime (GMT)
`endtime`	a POSIXct class specifying the endtime (GMT)
`metricName`	a character string containing one or more comma separated metric names
`constraint`	a character string containing value constraints
`url`	optional url of the MUSTANG measurements service

Details

The default MUSTANG measurement service when `url` is not specified is:

`http://service.iris.edu/mustang/measurements/1/query?`

Data returned from MUSTANG are converted into an R dataframe

The optional `constraint` parameter is used to add constraints to the query as defined in the hrefhttp://service.iris.edu/mustang/measurements/1MUSTANG measurements web serivce documentation. Any string passed in with the `constraint` parameter will be appended to the request url following an ampersand.

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

Value

A dataframe with the following columns:

`~metricName~, value, snclq, starttime, endtime, loadtime`

The `loadtime` column contains the time at which this record was loaded into the database.

The dataframe rows will be sorted by increasing starttime.

The structure of this dataframe is appropriate for use with the **ggplot2** plotting package.

Note

The database was originally populated with a version of this package that always assigned quality to be 'B'. Later versions obtained the quality from the miniSEED packet (typically 'M'). Because of this it is possible to have duplicate entries that only differ in the Q part of their snclq. To avoid double counting, when the webservice return contains two records whose only difference is the quality code portion of the of the snclq, only the record with the later loaddate will be used in the dataframe.

Author(s)

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

See Also

createBssUrl, getPsdMetrics

Examples

```
  ## Not run:
# Open a connection to IRIS DMC webservices (including the BSS)
iris <- new("IrisClient", debug=TRUE)

starttime <- as.POSIXct("2013-06-01", tz="GMT")
endtime <- starttime + 30*24*3600
metricName <- "sample_max,sample_min,sample_mean"

# Get the measurement dataframe
juneStats <- getSingleValueMetrics(iris,"IU","ANMO","00","BHZ",
                                   starttime,endtime,metricName)

# Simple ggplot2 plot
library(ggplot2)
p <- ggplot(juneStats, aes(x=starttime,y=value, color=as.factor(metricName))) +
     geom_step()
print(p)

## End(Not run)
```

`metricList2DFList` *Conver a metricList into a list of dataframes*

Description

The `metricList2DFList` function converts a list of `SingleValueMetrics` into a list of dataframes, one per named metric.

Usage

```
metricList2DFList(metricList)
```

Arguments

`metricList`	a list of `SingleValueMetric` objects

Details

Metrics functions return lists of `SingleValueMetric` objects. A long `metricList` may be built up by appending the results of different metrics functions or the same metrics function operating on different seismic signals. A metricList generated by any of the MUSTANG Rscripts can be stored as an `.RData` file and reloaded for examination.

A metricList may contain values for many different metrics. This function creates a separate dataframe for each `metricName` found in the `metricList`. As each dataframe is created, values associated with that metric are stored in a column named after the metric. Individual dataframes are stored in the returned list with their own name: `metricName_DF`.

Value

A character string with BSS formatted XML is returned.

Author(s)

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

See Also

`SingleValueMetric-class`, `metricList2Xml`

Examples

```
  ## Not run:
load('metricList_3756.RData')
dfList <- metricList2DFList(metricList)
names(dfList)
[1] "peak_correlation_DF" "peak_lag_DF"
corr <- dfList[["peak_correlation_DF"]]
names(corr)
```

```
[1] "snclq"                "starttime"            "endtime"              "attributeName"
[5] "attributeValueString" "quality_flag"         "peak_correlation"

## End(Not run)
```

`metricList2Xml` *Create XML for the BSS*

Description

The `metricList2Xml` function converts a list of `SingleValueMetrics` into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
metricList2Xml(metricList)
```

Arguments

`metricList`	a list of `SingleValueMetric` objects

Details

Metrics functions return lists of `SingleValueMetric` objects. A long `metricList` may be built up by appending the results of different metrics functions or the same metrics function operating on different seismic signals. These metrics can be submitted to the BSS in a standardized XML format. (see SingleValueMetric-class)

Value

A character string with BSS formatted XML is returned.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Apply a metric and show the results
metricList <- stateOfHealthMetric(st)
metricList <- append(metricList, basicStatsMetric(st))
bssXml <- metricList2Xml(metricList)
```

MultipleTimeValueMetric-class
Class "MultipleTimeValueMetric"

Description

A container for metrics consisting of a vector of POSIXct datetimes. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

new("MultipleTimeValueMetric", snclq, starttime, endtime, metricName, values)

Lists of MultipleTimeValueMetric objects are returned by various metrics functions in this package.

Slots

snclq: Object of class "character": SNCLQ identifier.

metricName: Object of class "character": Name of the metric.

elementName: Object of class "character": Name of the datetime element (default="t").

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

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

values: Object of class "POSIXct": Datetime values.

valueStrings: Object of class "character": String representations of the datetime values.

quality_flag: Object of class "numeric": Quality flag.

quality_flagString: Object of class "character": String representation of quality flag.

Methods

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

Note

The starttime and endtime slots are typically associated with the *user requested* times which may not match up with the starttime associated with the first Trace and the endtime associated with last Trace in the Stream object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
```

```
      <up_down_times>
        <t value="2012-02-10T00:00:00.000"/>
        <t value="2012-02-10T00:01:00.000"/>
        <t value="2012-02-10T00:02:00.000"/>
        <t value="2012-02-10T00:03:00.000"/>
      </up_down_times>
    </target>
  </date>
</measurements>
```

The quality_flag is an optional value avialable for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

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

See Also

upDownTimesMetric

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("meanMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# get the upDownTimes with a minimum signal length and minimum gap (secs)
upDownTimes <- getUpDownTimes(st, min_signal=30, min_gap=60)

# Create and return a MultipleTimeValue metric from the upDownTimes
m <- new("MultipleTimeValueMetric", snclq=snclq,
         starttime=starttime, endtime=endtime,
         metricName="up_down_times", values=upDownTimes)

# Show the results
show(m)
```

MultipleValueMetric-class
Class "MultipleValueMetric"

Description

A container for metrics consisting of a vector of numeric values. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

`new("MultipleValueMetric", snclq, starttime, endtime, metricName, values)`

Lists of `MultipleValueMetric` objects are returned by various metrics functions in this package.

Slots

`snclq`**:** Object of class "`character`": SNCLQ identifier.

`metricName`**:** Object of class "`character`": Name of the metric.

`elementName`**:** Object of class "`character`": Name of the element storing the metric value (default="`x`").

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

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

`values`**:** Object of class "`numeric`": Numeric values.

`valueStrings`**:** Object of class "`character`": String representations of the numeric values.

`quality_flag`**:** Object of class "`numeric`": Quality flag.

`quality_flagString`**:** Object of class "`character`": String representation of quality flag.

Methods

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

Note

The `starttime` and `endtime` slots are typically associated with the *user requested* times which may not match up with the `starttime` associated with the first `Trace` and the `endtime` associated with last `Trace` in the `Stream` object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
      <EXAMPLE>
```

```
        <x value="1"/>
        <x value="2"/>
        <x value="3"/>
        <x value="4"/>
      </EXAMPLE>
    </target>
  </date>
</measurements>
```

The quality_flag is an optional value avialable for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

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

PSDMetric *Power Spectral Density of a signal*

Description

The PSDMetric() function performs spectral analysis on a seismic signal and returns 'PSD' metrics with discretized spectral components as well as other metrics based on PSD's

Usage

```
PSDMetric(st,
          expLoPeriod=4/(st@traces[[1]]@stats@sampling_rate),
          expHiPeriod=100,
          linLoPeriod=16/(st@traces[[1]]@stats@sampling_rate),
          linHiPeriod=50)
```

Arguments

st	a Stream object containing a seismic signal
expLoPeriod	the low end of the period band use for calculating the exponential dead channel detector
expHiPeriod	the high end of the period band use for calculating the exponential dead channel detector
linLoPeriod	the low end of the period band use for calculating the linear dead channel detector
linHiPeriod	the high end of the period band use for calculating the linear dead channel detector

Details

This metric calculates average power spectra for a seismic signal as described in the McNamara paper. See the `McNamaraPSD` method of `Stream` objects in the **seismic** package for details.

Uncorrected spectral density values are returned in `spectrumMetricList` in units of dB.

Other metrics calculated from the PSDs are returned in `svMetricList`. These metrics are:

pct_above_nhnm – "percent above New High Noise Model" Percentage of PSD values that are above the New High Noise Model for their frequency. This value is calculated over the entire time period.

pct_below_nlnm – "percent below New Low Noise Model" Percentage of PSD values that are below the New Low Noise Model for their frequency. This value is calculated over the entire time period.

dead_channel_exp – "dead channel metric - exponential fit" A "dead channel" metric is calculated from the mean of all the PSDs generated. (Typically 47 for a 24 hour period.) Values of the PSD mean line over the band (expLoPeriod:expHiPeriod) are fit to an exponential. The `dead_channel_exp` metric is the standard deviation of the fit residuals. Lower numbers indicate a better fit and a higher likelihood that the mean PSD is exponential – an indication of a "dead channel".

dead_channel_lin – "dead channel metric - linear fit" A "dead channel" metric is calculated from the mean of all the PSDs generated. (Typically 47 for a 24 hour period.) Values of the PSD mean line over the band (linLoPeriod:linHiPeriod) are fit to a line. The `dead_channel_lin` metric is the standard deviation of the fit residuals. Lower numbers indicate a better fit and a higher likelihood that the mean PSD is linear – an indication of a "dead channel".

Value

A list of lists is returned containing:

- `spectrumMetricList` = list of `SpectrumMetric` objects
- `svMetricList` = list of `SingleValueMetric` objects:
  - `pct_above_nhnm`
  - `pct_below_nlnm`
  - `dead_channel_exp`
  - `dead_channel_lin`

Author(s)

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

References

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

See Also

`SpectrumMetric` `SingleValueMetric`

Examples

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

# NOTE:  The following trace has 1.728 million points.
# NOTE:  Downloading and calculating PSD may take a few seconds.
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)

# Calculate the PSD metric and show the SingleValueMetric results
listOfLists <- PSDMetric(st)
svMetricList <- listOfLists[['svMetricList']]

dummy <- lapply(svMetricList, show)

## End(Not run)
```

`saveMetricList` *Save a MetricList as RData or XML*

Description

The saveMetricList() function allows metrcis to be saved as either .RData files or as XML. The XML format is the same as that used by the IRIS DMC MUSTANG database for metric submission.

Usage

```
saveMetricList(metricList, id=Sys.getpid(), rdata=FALSE)
```

Arguments

`metricList`	list of SingleValueMetric objects
`id`	ID to be used when generating output files
`rdata`	optional flag to save the incoming `metricList` as a .RData file

Details

The `saveMetricList` function saves a list of `SingleValueMetrics` as a .RData binary file or converts the list into the XML format expected by the MUSTANG database submission process. This XML format is human readable and can be used to spot check results of metrics calculations.

Value

The automatically generated filename is returned invisibly.

Author(s)

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

See Also

SingleValueMetric-class, metricList2Xml, getMetricsXml, getBssMetricList,

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Apply a metric and show the results
metricList <- stateOfHealthMetric(st)
metricList <- append(metricList, basicStatsMetric(st))
saveMetricList(metricList,id='AK.PIN..VEA')

## End(Not run)
```

SingleValueMetric-class

Class "SingleValueMetric"

Description

A container for metrics results and associated metadata. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

`new("SingleValueMetric", snclq, starttime, endtime, metricName, value)`

Lists of `SingleValueMetric` objects are returned by various metrics functions in this package.

Slots

`snclq`: Object of class `"character"`: SNCLQ identifier.

`metricName`: Object of class `"character"`: Name of the metric.

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

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

`valueName`: Object of class `"character"`: Name of the XML value identifier (default=`"value"`).

`value`: Object of class `"numeric"`: Metric value.

`valueString`: Object of class `"character"`: String representation of the metric value.

`quality_flag`: Object of class `"numeric"`: Quality flag.

`quality_flagString`: Object of class `"character"`: String representation of quality flag.

`attributeName`: Object of class `"character"`: Name of one or more optional attributes.

`attributeValueString`: Object of class `"character"`: String representation of one or more attribute values.

Methods

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

Note

The `starttime` and `endtime` slots are typically associated with the *user requested* times which may not match up with the `starttime` associated with the first `Trace` and the `endtime` associated with last `Trace` in the `Stream` object being analyzed. This ensures that metrics results for a single time period but covering many stations or channels will have the same date range and improves performance of the BSS which expects XML of the following form:

```
<measurements>
  <date start='2012-02-10T00:00:00.000' end='2012-02-10T09:20:00.000'>
    <target snclq='N.S.L.C1.Q'>
      <example value='1.0'/>
    </target>
    <target snclq='N.S.L.C2.Q'>
      <example value='2.0'/>
    </target>
    <target snclq='N.S.L.C3.Q'>
      <example value='3.0'/>
    </target>
  </date>
  </date>
</measurements>
```

The `quality_flag` is an optional value available for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

For an IRIS/DMC specific example, the `station_completeness` metric obtains a list of available channels for a station from the availability web service and compares this list with the list of `percent_availability` metrics for this station stored in the MUSTANG BSS. In the case of the `station_completeness` metric, the `quality_flag` is set to the number of channels that should be available but for whom no `percent_availability` measure is obtained form the BSS.

The `attributeName` and `attributeValueString` slots can be used to store additional attributes associated with a metric values. For example, the `max_stalta` value for a seismic trace can be calculated and a metric can be created that contains this value and another attribute with a string representation of the time at which this maximum occurred.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Apply a metric and show the results
metricList <- basicStatsMetric(st)
show(metricList[[1]])
```

SNRMetric *Signal to Noise Ratio*

Description

The SNRMetric() function calculates the Signal-to-Noise Ratio of a seismic trace by one of several named algorithms.

Usage

```
SNRMetric(st, algorithm, windowSecs)
```

Arguments

st	a Stream object containing a seismic signal
algorithm	a named algorithm to use for calculating SNR (default="splitWindow")
windowSecs	width (secs) of the full window used in SNR calculations (default=60)

Details

Seismic signals in the Stream must be without gaps, *i.e.* contained within a single Trace.

algorithm="splitWindow"

This algorithm uses the midpoint of the seismic signal as the border between noise to the left of the midpoint and signal to the right. The value for signal-to-noise is just the rmsVariance calculated for windowSecs/2 seconds of data to the right of the midpoint divided by the rmsVariance for windowSecs/2 seconds of data to the left of the midpoint.

No other algorithms have been vetted at this point.

Value

A list with a single SingleValueMetric object is returned.

Author(s)

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

Examples

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

# Get an hour long waveform centered on a big quake
starttime <- as.POSIXct("2010-02-27 06:16:15",tz="GMT")
endtime <- as.POSIXct("2010-02-27 07:16:15",tz="GMT")
st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
tr <- st@traces[[1]]

# Calculate the SNR metric and show the results
metricList <- SNRMetric(st)
dummy <- lapply(metricList, show)
```

SpectrumMetric-class *Class* "SpectrumMetric"

Description

A container for metrics consisting of discrete spectra. This information is used to create XML that is then submitted to the MUSTANG Backend Storage System (BSS).

Objects from the Class

Objects can be created by calls of the form:

```
new("SpectrumMetric", snclq, starttime, endtime, metricName, freqs, amps, phases)
```

Slots

snclq: Object of class "character": SNCLQ identifier.

metricName: Object of class "character": Name of the metric.

elementName: Object of class "character": Name of the datetime element (default="t").

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

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

freqs: Object of class "numeric": Frequency values.

freqStrings: Object of class "character": String representations of the frequency values.

amps: Object of class "numeric": Amplitude values.

ampStrings: Object of class "character": String representations of the amplitude values.

phases: Object of class "numeric": Phase values.

phaseStrings: Object of class "character": String representations of the phase values.

quality_flag: Object of class "numeric": Quality flag.

quality_flagString: Object of class "character": String representation of quality flag.

Methods

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

Note

The quality_flag is an optional value avialable for storing information related to the processing of a particular metric. Its meaning will vary from metric to metric.

Author(s)

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

---

spectrumMetric2Xml *Convert a SpecturmMetric into XML for the BSS*

Description

The spectrumMetric2Xml function converts a SpectrumMetric into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
spectrumMetric2Xml(metricList)
```

Arguments

metricList	a list of SpectrumMetric objects

Value

A character string with BSS formatted XML is returned.

Author(s)

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

Examples

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

# NOTE:  The following trace has 1.728 million points.
# NOTE:  Downloading and calculating PSD may take a while.
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)

# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("PSDMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# Calculate and plot the Power Spectral Density
psd <- PSD(st)

# Create a Spectrum metric
# TODO:  total revamp of PSD calculations to match Manochehr's needs
phases <- rep(0,length(psd$freq))
m <- new("SpectrumMetric", snclq=snclq, starttime=starttime, endtime=endtime,
          metricName="psd", freqs=psd$freq, amps=psd$spec, phases=psd$phase)

# Show the XML version of the metric
bssXml <- spectrumMetric2Xml(m)
cat(bssXml)

## End(Not run)
```

`spikesMetric` *Find spikes using a rolling Hampel filter*

Description

The spikesMetric() function determines the number of spikes in a seismic `Stream`.

Usage

```
spikesMetric(st, windowSize, thresholdMin, selectivity)
```

Arguments

st	a Stream object containing a seismic signal
windowSize	The window size to roll over (default=41)
thresholdMin	Minimum value for outlier detection (default=6.0)
selectivity	Numeric factor [0-1] used in determining outliers (default=0.4)

Details

This function uses the output of the findOutliers function in the **seismicRoll** package to calculate the number of 'spikes' containing outliers.

The thresholdMin level is similar to a sigma value for normally distributed data. Hampel filter values above 6.0 indicate a data value that is extremely unlikely to be part of a normal distribution (~ 1/500 million) and therefore very likely to be an outlier. By choosing a relatively large value for thresholdMin we make it less likely that we will generate false positives.

The selectivity is a value between 0 and 1 and is used to generate an appropriate threshold for outlier detection based on the statistics of the incoming data. A lower value for selectivity will result in more outliers while a value closer to 1.0 will result in fewer.

The total count of spikes reflects the number of outlier data points that are separated by at least one non-outlier data point. Each individual spike may contain more than one data point.

Value

A list of SingleValueMetric objects is returned.

Note

The thresholdMin parameter is sensitive to the data sampling rate. The default value of 6.0 seems to work well with sampling rates of 10 Hz or higher ('B..' or 'V..' channels). For 'L..' channels with a sampling rate of 1 Hz it seems that thresholdMin=12.0 is more appropriate.

More testing of spiky signals at different resolutions is needed.

See the **seismicRoll** package for documentation on the findOutliers() function.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2013-01-03 15:00:00", tz="GMT")
endtime <- starttime + 3600 * 3
st <- getDataselect(iris,"IU","RAO","10","BHZ",starttime,endtime)

# Calculate the gaps metrics and show the results
```

```
metricList <- spikesMetric(st)
dummy <- lapply(metricList, show)
```

STALTAMetric *Maximum STA/LTA of a signal*

Description

The STALTAMetric() function calculates the maximum of STA/LTA over the incoming seismic signal.

Usage

```
STALTAMetric(st, staSecs, ltaSecs, increment, algorithm)
```

Arguments

st	a Stream object containing a seismic signal
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")
increment	increment used when sliding the averaging windows to the next location (default=1)

Details

Currently supported algorithms include:

- "classic_RR"
- "classic_LR"
- "EarleAndShearer_envelope"

This metric applies the STALTA method of Trace objects to every Trace in st with the following parameter settings:

- demean=TRUE
- detrend=TRUE
- taper=0.0

The final metric value is the maximum STALTA value found in any Trace in this Stream.

Further details are given in the documentation for STALTA.Trace().

Value

A list with a single SingleValueMetric object is returned. The metric name is max_stalta.

Note

The STALTA method of Trace objects returns a numeric vector of STA/LTA values that has the same length as the signal data. This is a moderately time consuming operation. By comparison, finding the maximum value of this vector of STA/LTA values is very fast.

Author(s)

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

References

First break picking (Wikipedia)

Automatic time-picking of first arrivals on noisy microseismic data (Wong et. al. 2009)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-02-12",tz="GMT")
endtime <- as.POSIXct("2012-02-13",tz="GMT")
st <- getDataselect(iris,"AK","GHO","","BHN",starttime,endtime)

# Calculate the STA/LTA metric and show the results
metricList <- STALTAMetric(st)
dummy <- lapply(metricList, show)

## End(Not run)
```

---

stateOfHealthMetric *State of Health metrics*

Description

The stateOfHealthMetric function extracts accumulated miniSEED quality flags and a measure of timing quality associated with the incoming seismic signal.

Usage

```
stateOfHealthMetric(st)
```

Arguments

st	a Stream object containing a seismic signal

Details

The miniSEED flags and timing_qual values are described in the SEED manual (http://www.fdsn.org/seed_manual/SEEDManual_V2.4.pdf).

Each Stream object contains "accumulators" with counts of the number of times each bit flag was set during the parsing of a miniSEED file. Metrics are reported for a subset of these flags as show in the code snippet below:

```
# act_flags
calibration_signal <- st@act_flags[1]
timing_correction <- st@act_flags[2]
event_begin <- st@act_flags[3]
event_end <- st@act_flags[4]
event_in_progress <- st@act_flags[7]

# io_flags
clock_locked <- st@io_flags[6]

# dq_flags
amplifier_saturation <- st@dq_flags[1]
digitizer_clipping <- st@dq_flags[2]
spikes <- st@dq_flags[3]
glitches <- st@dq_flags[4]
missing_padded_data <- st@dq_flags[5]
telemetry_sync_error <- st@dq_flags[6]
digital_filter_charging <- st@dq_flags[7]
```

An additional "timing quality" metric gives the average value for the timing_qual value associated with each block of miniSEED data.

Value

A list of SingleValueMetric objects is returned.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Generate State of Health metrics and show the results
metricList <- stateOfHealthMetric(st)
dummy <- lapply(metricList, show)
```

timesMetric2Xml *Create XML for the BSS*

Description

The timesMetric2Xml function converts a MultipleTimeValueMetric into an XML structure appropriate for submitting to the MUSTANG Backend Storage System (BSS).

Usage

```
timesMetric2Xml(metric)
```

Arguments

metric	a MultipleTimeValueMetric object

Value

A character string with BSS formatted XML is returned.

Author(s)

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

Examples

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

# Get the waveform
starttime <- as.POSIXct("2012-01-24", tz="GMT")
endtime <- as.POSIXct("2012-01-25", tz="GMT")
st <- getDataselect(iris,"AK","PIN","","VEA",starttime,endtime)

# Make sure we're working with a single snclq
unique_ids <- uniqueIds(st)
if (length(unique_ids) > 1) {
  stop(paste("meanMetric: Stream has",unique_ids,"unique identifiers"))
}
snclq <- unique_ids[1]

# get the upDownTimes with a minimum signal length and minimum gap (secs)
upDownTimes <- getUpDownTimes(st, min_signal=30, min_gap=60)

# Create and return a MultipleTimeValue metric from the upDownTimes
m <- new("MultipleTimeValueMetric", snclq=snclq, starttime=starttime,
         endtime=endtime, metricName="up_down_times", values=upDownTimes)

# Show the XML version of the metric
```

```
bssXml <- timesMetric2Xml(m)
cat(bssXml)
```

`transferFunctionMetric` *Cross-spectral comparison*

Description

The transferFunctionMetric() function calculates metrics that assess the relationship between two SNCLs with the same network, station and channel but separate locations. When seismometers are working properly, the transfer function amplitude and phase will match similar values calculated from the instrument responses.

This function calculates the transfer function from data in the incoming streams. Response information is then obtained from the evalresp web service.

Usage

```
transferFunctionMetric(st1, st2)
```

Arguments

`st1`	a `Stream` object containing a seismic signal
`st2`	a `Stream` object containing a seismic signal

Details

Details of the algorithm are as follows

```
# compute complex cross-spectrum of traces x and y ==> Pxx, Pxy, Pyy
# calculate transfer function values:
#     Txy(f) = Pxy(f) / Pxx(f)
#     dataGain <- Mod(Txy)
#     dataPhase <- Arg(Txy)
#
# calculate avgDataGain and avgDataPhase values for periods of 5-7s
#
# calculate the corresponding response amplitude ratio and phase difference:
#     request responses for x and y
#     respGain = respGainy(f) / respGainx(f)
#     respPhase = respPhasey(f) - respPhasex(f)
#
# calculate avgRespGain and avgRespPhase values for periods of 5-7s
#
# calculate metrics:
#     gain_ratio = avgDataGain / avgRespGain
#     hase_diff = avgDataPhase - avgRespPhase
#     ms_coherence = |Pxy|^2 / (Pxx*Pyy)
```

Value

A list with a single SingleValueMetric object is returned. The metric name is transfer_function and it has three attributes:

- gain_ratio – reasonableness of cross-spectral amplitude between st1 and st2
- phase_diff – reasonableness of cross-spectral phase between st1 and st2
- ms_coherence – mean square coherence between st1 and st2

These values can be interpreted as follows:

Whenever ms_coherence ~= 1.0, properly functioning seismometers should have:

- gain_raio ~= 1.0
- phase_diff < 10.0 (degrees)

Note

Seismic streams passed to transferFunctionMetric() must have the same network, station and channel and must cover the same time range. The two channels should also have values of azimuth and dip within five degrees of each other. If sampling rates differ, the stream with the higher sampling rate will be decimated to match the lower sampling rate.

The metricList generated for these two-channel metrics will have a SNCL code of the form: N.S.L1:L2.C.Q.

Author(s)

Jonathan Callahan <jonathan.s.callahan@gmail.com> (R code), Mary Templeton <met@iris.washington.edu> (algorithm)

Examples

```
# Create a new IrisClient
iris <- new("IrisClient", debug=TRUE)

# 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)

# Calculate metrics
metricList <- transferFunctionMetric(st1,st2)
print(metricList)
```

upDownTimesMetric *Up/down times for a channel*

Description

The upDownTimesMetric() function determines the times at which data collection starts and stops within a seismic Stream.

Usage

```
upDownTimesMetric(st, min_signal, min_gap)
```

Arguments

st	a Stream object containing a seismic signal
min_signal	minimum duration of a Trace in seconds (default=30)
min_gap	minimum gap in seconds (default=60)

Details

This function uses the output of the getUpDownTimes method of Stream objects.

Value

A list with a single MultipleTimeValueMetric object is returned.

Note

See the seismic package for documentation on Stream objects and the getDataselect method.

Author(s)

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

See Also

getUpDownTimes

Examples

```
# Open a connection to IRIS DMC 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","","VEA",starttime,endtime)
```

```
# Create the upDownTimesMetric, ignoring Traces < 3 minutes and gaps of < 5 minutes
metricList <- upDownTimesMetric(st, min_signal=180, min_gap=300)
```

Index