Introduction to the `IRISSeismic` Package for R

Jonathan Callahan* Mazama Science
Robert Casey† IRIS DMC
Mary Templeton‡ IRIS DMC

August 2015

Contents

*jonathan@mazamascience.com
†rob@iris.washington.edu
‡met@iris.washington.edu

1 Introduction

The `IRISSeismic` package for seismic data analysis is being developed by Mazama Science for the IRIS DMC (Incorporated Research Institutions for Seismology - Data Management Center). This development is part of the **MUSTANG** project for automated QC of seismic data.

The goal of this package is to make it easy to obtain and work with data from IRIS DMC web services. This introduction will demonstrate some of the core functionality of the `IRISSeismic` package and how it can be used in interactive sessions. Detailed information about object properties and function arguments can be found in the package documentation.

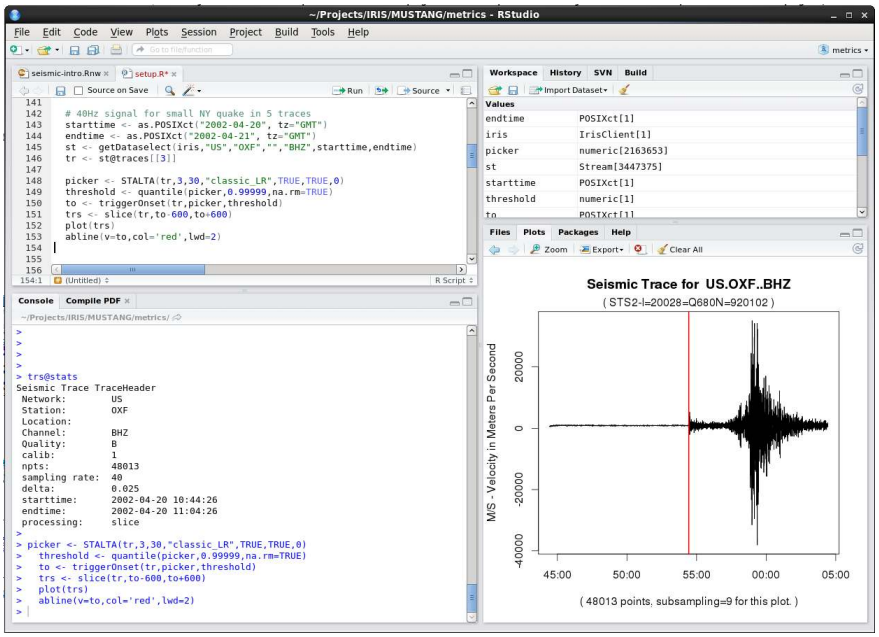
The core objects in this package, especially `Trace` and `Stream` objects, borrow heavily from concepts and features found in the Python ObsPy package. References to specific **ObsPy** classes can be found in the source code.

2 Getting Started

For those who are not used to working with `R`, the Using R series of blog posts offers tips on how to get started and includes links to other introductory documentation.

2.1 RStudio

Users of the `IRISSeismic` package are encouraged to first download and install the RStudio integrated development environment for **R**. Newcomers to `R` will find **RStudio** a much friendlier environment in which to work.



2.2 First example

Once you have an **R** environment up and running, the first step is to load the `IRISSeismic` package. Then you can create a new `IrisClient` object that will be responsible for all subsequent communication with DMC provided web services.

```
> library(IRISSeismic)
> iris <- new("IrisClient")
```

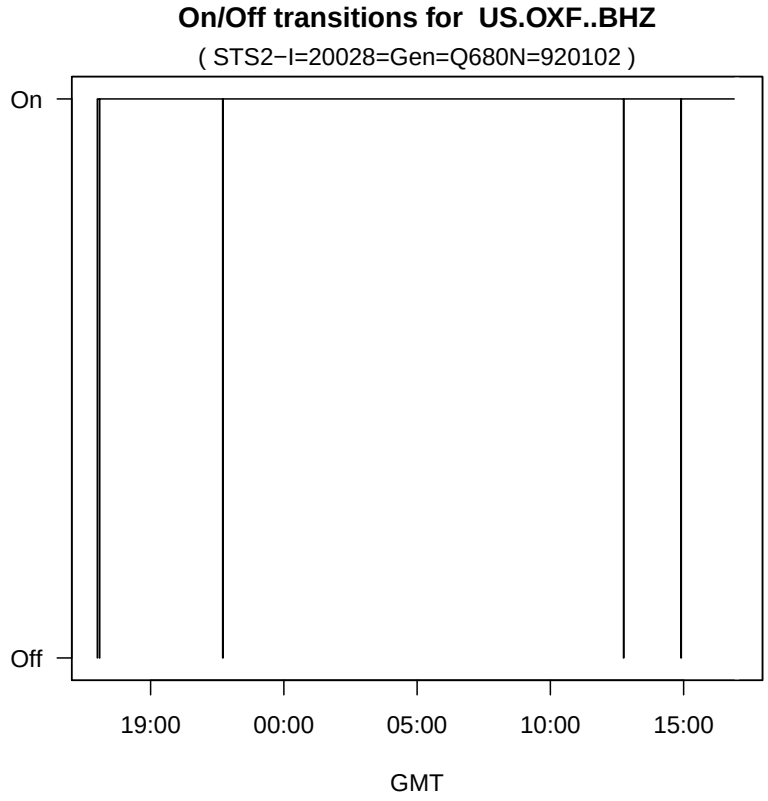
In order to get data from one of the IRIS DMC web services we must specify all the information needed to create a webservice request: `network, station, location, channel, starttime, endtime`. Each unique combination of these elements is known as a *SNCL*. These elements are passed to the `getDataselect()` method of the `IrisClient` as a series of character strings except for the times which are of type `POSIXct`. The user is responsible for creating datetime objects of class `POSIXct`.

The first three commands in the following code chunk use the `IrisClient` object to communicate with web services and return a `Stream` object full of data from the IRIS DMC. The fourth line checks to see how many distinct chunks of seismic data exist. The last line passes this `Stream` object to a function that will plot the times at which this channel was collecting data.

```
> starttime <- as.POSIXct("2002-04-20", tz="GMT")
> endtime <- as.POSIXct("2002-04-21", tz="GMT")
> st <- getDataselect(iris,"US","OXF","","BHZ",starttime,endtime)
> length(st@traces)

[1] 5

> plotUpDownTimes(st, min_signal=1, min_gap=1)
```



This station had a few minor data dropouts, causing the data to be broken up into several separate signals that the `IRISSeismic` package stores in `Trace` objects.

We can get more information on the gaps between traces with the `getGaps()` function. The duration (secs) of gaps between traces is displayed along with the number of samples that were missed during the gap.

```
> getGaps(st)

$gaps
[1]  0.0000 58.7750 57.0749 47.5750 52.1750  0.0000

$nsamples
[1]    0 2351 2283 1903 2087    0
```

Next we can examine various statistics for each individual trace with the following `parallel-` functions.

```
> parallelLength(st)

[1]   10733  663953 2163653  308769  300267

> parallelMax(st)

[1]  1218  1356 38406  6139  1305

> parallelSd(st)

[1] 101.14186  97.03080 484.53911 135.00670  93.05572
```

It looks like the third trace, with a larger maximum and standard deviation, might have a signal. Metadata for this trace is stored in the `stats` slot of the `Trace` object.

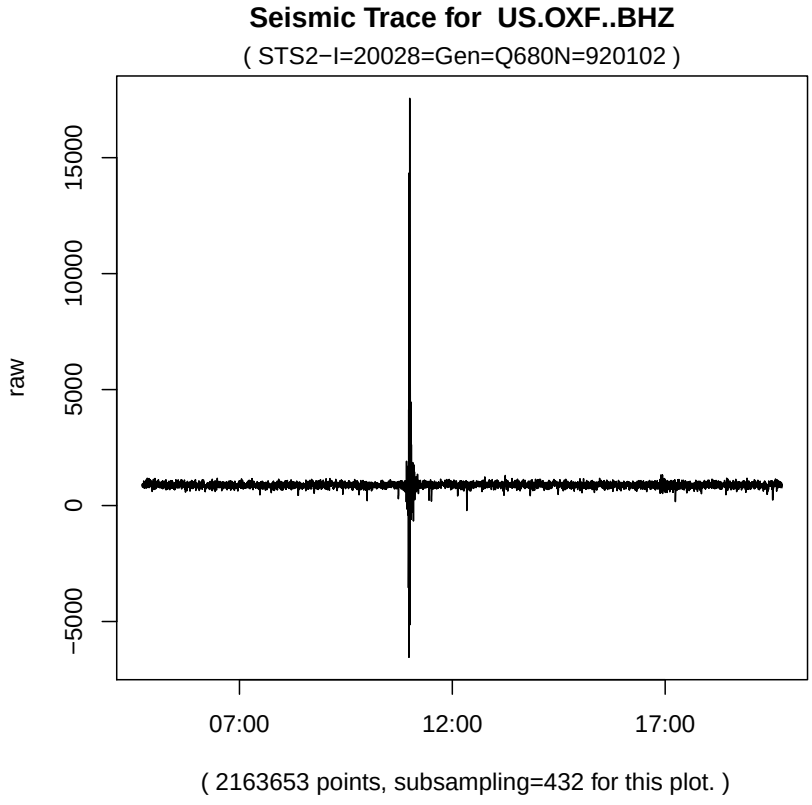
```
> tr <- st@traces[[3]]
> tr@stats

Seismic Trace TraceHeader
 Network:        US
 Station:        OXF
 Location:
 Channel:        BHZ
 Quality:        M
 calib:          1
 npts:           2163653
 sampling rate:  40
 delta:          0.025
```

```
starttime:      2002-04-20 04:43:03
endtime:        2002-04-20 19:44:34
latitude:       34.5118
longitude:      -89.4092
elevation:      101
depth:          0
azimuth:        0
dip:            -90
processing:
```

Finally, we can look at the seismic signal with the `plot` method.

```
> plot(tr)
```



This small seismic signal was recorded in Oxford, Mississippi and is from a quake that occurred in New York state

Note: By default, data are subsampled before plotting to *greatly!* improve plotting speed. You can sometimes improve the appearance of a plot by reducing the amount of subsampling used. The `plot` method accepts a `subsampling` parameter to specify this.

2.3 Understanding `Stream` and `Trace` objects

In order to work effectively with the `IRISSeismic` package you must first understand the structure of the new `S4` objects it defines. The package documentation gives a full description of each object but we can also interrogate them using the `slotNames()` function.

```
> slotNames(st)
```

```
[1] "url"                "requestedStarttime" "requestedEndtime"
[4] "act_flags"          "io_flags"           "dq_flags"
[7] "timing_qual"        "traces"
```

The `Stream` object has the following *slots* (aka *properties* or *attributes*):

- `url` – full web services URL used to obtain data
- `requestedStarttime` – POSIXct datetime of the requested start
- `requestedEndtime` – POSIXct datetime of the requested end
- `act_flags` – act flags from the miniSEED record
- `io_flags` – act flags from the miniSEED record
- `dq_flags` – act flags from the miniSEED record
- `timing_qual` – timing quality from the miniSEED record
- `traces` – list of `Trace` objects

When in doubt about what a particular *slot* contains, it is always a good idea to ask what type of object it is.

```
> class(st@url)
```

```
[1] "character"
```

```
> class(st@requestedStarttime)
```

```
[1] "POSIXct" "POSIXt"
```

```
> class(st@traces)
```

```
[1] "list"
```

The next code chunk examines the first `Trace` in our `Stream`.

> **Note:** `R` uses double square brackets, `[[...]]` to access list items.

```
> slotNames(st@traces[[1]])
```

```
[1] "id"                    "stats"                 "Sensor"
[4] "InstrumentSensitivity" "SensitivityFrequency"  "InputUnits"
[7] "data"
```

The `Trace` object has the following slots:

- `id` – SNCLQ identifier of the form "US.OXF..BHZ.B"
- `stats` – `TraceHeader` object (metadata from the trace)
- `Sensor` – instrument equipment name
- `InstrumentSensitivity` – instrument total sensitivity (stage 0 gain)
- `SensitivityFrequency` – the frequency (Hz) at which the `InstrumentSensitivity` is correct
- `InputUnits` – instrument data acquisition input units
- `data` – vector of `numeric` data (the actual signal)

The `TraceHeader` metadata and the actual signal come from the dataselect webservice. The instrument metadata are obtained from the station webservice.

2.4 Be careful with times

Time stamps associated with seismic data should be given as "Universal" or "GMT" times. When specifying times to be used with methods of the `IRISSeismic` package you must be careful to specify the timezone as R assumes the local timezone by default.

Also, R assumes that datetime strings are formatted with a space separating date and time as opposed to the ISO 8601 'T' separator. If an ISO 8601 character string is provided without specific formatting instructions, the time portion of the string will be lost *without any warning!* So it is very important to be careful and consistent if you write code that converts ASCII strings into times.

A few examples will demonstrate the issues:

```
> as.POSIXct("2010-02-27", tz="GMT") # good
```

```
[1] "2010-02-27 GMT"
```

```
> as.POSIXct("2010-02-27 04:00:00", tz="GMT") # good
```

```
[1] "2010-02-27 04:00:00 GMT"
```

```
> as.POSIXct("2010-02-27T04:00:00", tz="GMT",
+            format="%Y-%m-%dT%H:%M:%OS") # good
```

```
[1] "2010-02-27 04:00:00 GMT"
```

```
> as.POSIXct("2010-02-27") # BAD -- no timezone
```

```
[1] "2010-02-27 PST"
```

```
> as.POSIXct("2010-02-27T04:00:00", tz="GMT") # BAD -- no formatting
```

```
[1] "2010-02-27 GMT"
```

3 Example Operations

The example at the beginning of this vignette already demonstrated how to obtain seismic data from DMC web services, how to learn about the number

and size of individual traces within the requested time range and how to generate a first plot of the seismic signal. This section will introduce more use cases that delve further into the capabilities of the `IRISSeismic` package. For complete details on available functions, please see the package documentation.

```
> help("IRISSeismic",package="IRISSeismic")
```

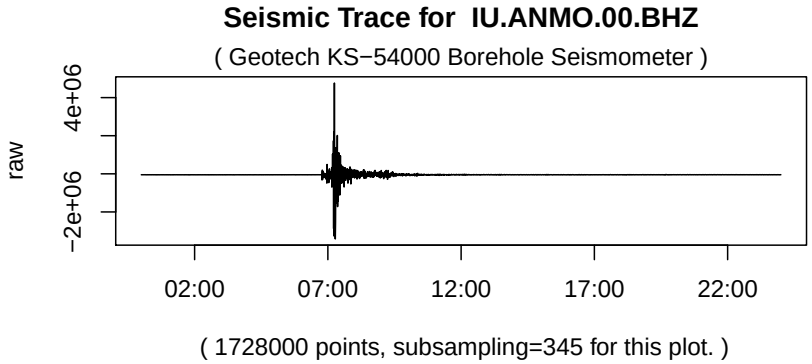
3.1 Closer examination of a seismic signal

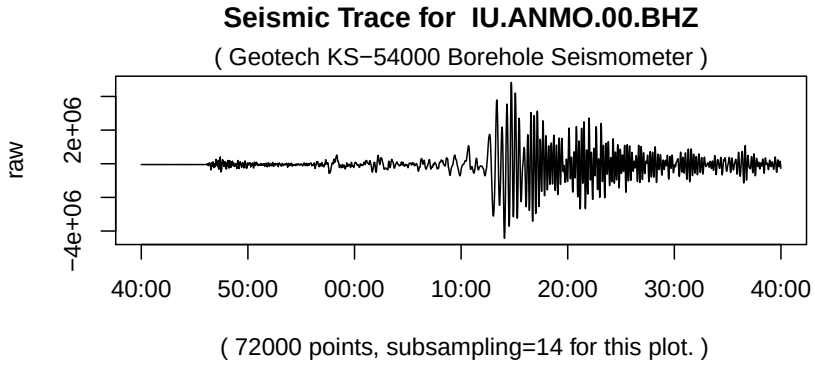
Once seismic data are in memory, performing mathematical analysis on those data can be very fast. All mathematical operations are performed on every data point. But plotting can still be a slow process.

> **Note:** The `plot()` method of `Stream` objects deals with gaps by first calling `mergeTraces()` to fill all gaps with missing values (`NA`). Then the single, merged trace is plotted with the `plot()` method for `Trace` objects. Any gaps of a significant size will be now visible in the resulting plot.

By default, the `plot()` method of `Trace` and `Stream` objects subsamples the data so that approximately 5,000 points are used in the plot. This dramatically speeds up plotting. One of the first things you will want to do with a full day's worth of seismic signal is clip it to a region of interest. One way to do that would be to modify the `starttime` and `endtime` parameters to `getDataselect` and then make a data request covering a shorter period of time. A simpler technique, if the signal is already in memory, is to use the `slice()` method.

```
> starttime <- as.POSIXct("2010-02-27", tz="GMT")
> endtime <- as.POSIXct("2010-02-28", tz="GMT")
> st <- getDataselect(iris,"IU","ANMO","00","BHZ",starttime,endtime)
> start2 <- as.POSIXct("2010-02-27 06:40:00", tz="GMT")
> end2 <- as.POSIXct("2010-02-27 07:40:00", tz="GMT")
> tr1 <- st@traces[[1]]
> tr2 <- slice(tr1, start2, end2)
> layout(matrix(seq(2)))        # layout a 2x1 matrix
> plot(tr1)                     # top
> plot(tr2)                     # bottom
> layout(1)                     # restore original layout
```





3.2 Detecting events with STA/LTA

Access to triggering algorithms for detecting events is provided by the `STALTA()` method of `Trace` objects. (*cf* A Comparison of Select Trigger Algorithms for Automated Global Seismic Phase and Event Detection) The `STALTA()` method has the following arguments and defaults:

- `x` – `Trace` being analyzed
- `staSecs` – size of the short window in secs
- `ltaSecs` – size of the long window in secs
- `algorithm` – named algorithm (default="classic_LR")
- `demean` – should the signal have the mean removed (default=`TRUE`)

- `detrend` – should the signal have the trend removed (default=`TRUE`)
- `taper` – portion of the seismic signal to be tapered at each end (default=0.0)
- `increment` – integer increment to use when sliding the averaging windows to the next location (default=1)

The `STALTA()` method returns a *picker*, a vector of numeric values, one for every value in the `Trace@data` slot. Note that this is a fairly compute-intensive operation. This picker can then be used with the `triggerOnset()` function to return the **approximate** start of the seismic signal.

We'll test this with our original seismic signal.

```
> starttime <- as.POSIXct("2002-04-20", tz="GMT")
> endtime <- as.POSIXct("2002-04-21", tz="GMT")
> st <- getDataselect(iris,"US","OXF","","BHZ",starttime,endtime)
> tr <- st@traces[[3]]
> picker <- STALTA(tr,3,30)
> threshold <- quantile(picker,0.99999,na.rm=TRUE)
> to <- triggerOnset(tr,picker,threshold)
```

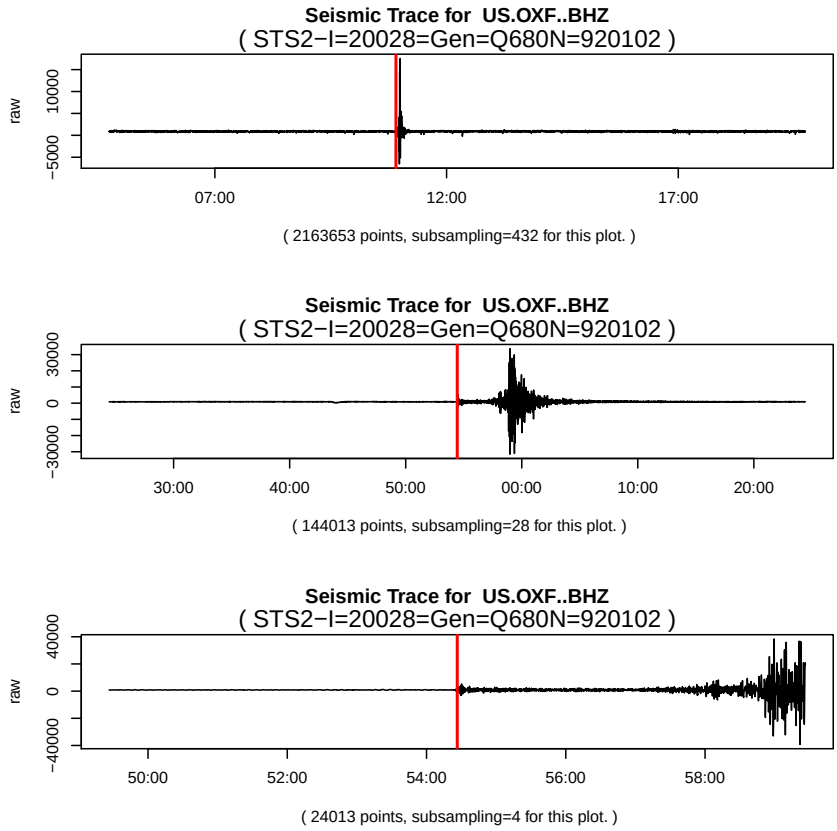
> **NOTE:** The `STALTA()` method is intended to be used for crude, automatic event detection, not precise determination of signal arrival. Optimal values for the arguments to the `STALTA()` method will depend on the details of the seismic signal.

The `eventWindow()` method allows you to focus on the region identified by the picker by automatically finding the trigger onset time and then slicing out the region of the trace centered on that time. This method has the following arguments and defaults:

- `x` – `Trace` being analyzed
- `picker` – picker returned by `STALTA()`
- `threshold` – threshold value as used in `triggerOnset()`
- `windowSecs` – total window size (secs)

```
> layout(matrix(seq(3)))        # layout a 3x1 matrix
> closeup1 <- eventWindow(tr,picker,threshold,3600)
> closeup2 <- eventWindow(tr,picker,threshold,600)
> plot(tr)
```

```
> abline(v=to, col='red', lwd=2)
> plot(closeup1)
> abline(v=to, col='red', lwd=2)
> plot(closeup2)
> abline(v=to, col='red', lwd=2)
> layout(1)                       # restore original layout
```



3.3 Data availability

The `IrisClient` also provides functionality for interacting with other web services at the DMC. The `getAvailability()` method allows users to query what SNCLs are available, obtaining that information from the station webservice.

Information is returned as a dataframe containing all the information returned by ws-availability. Standard DMC webservice wildcards can be used as in the example below which tells us what other 'B' channels are available at our station

of interest during the time of the big quake above.

```
> starttime <- as.POSIXct("2010-02-27", tz="GMT")
> endtime <- as.POSIXct("2010-02-28", tz="GMT")
> availability <- getAvailability(iris,"IU","ANMO","*","B??",starttime,endtime)
> availability

  network station location channel latitude longitude elevation depth azimuth
1      IU    ANMO       00     BH1 34.94598 -106.4571    1671.0 145.0     328
2      IU    ANMO       00     BH2 34.94598 -106.4571    1671.0 145.0      58
3      IU    ANMO       00     BHZ 34.94598 -106.4571    1671.0 145.0       0
4      IU    ANMO       10     BH1 34.94591 -106.4571    1767.2  48.8      64
5      IU    ANMO       10     BH2 34.94591 -106.4571    1767.2  48.8     154
6      IU    ANMO       10     BHZ 34.94591 -106.4571    1767.2  48.8       0
  dip                            instrument       scale scalefreq scaleunits
1   0 Geotech KS-54000 Borehole Seismometer  3456610000      0.02        M/S
2   0 Geotech KS-54000 Borehole Seismometer  3344370000      0.02        M/S
3 -90 Geotech KS-54000 Borehole Seismometer  3275080000      0.02        M/S
4   0  Guralp CMG3-T Seismometer (borehole) 32805600000      0.02        M/S
5   0  Guralp CMG3-T Seismometer (borehole) 32655000000      0.02        M/S
6 -90  Guralp CMG3-T Seismometer (borehole) 33067200000      0.02        M/S
  samplerate           starttime             endtime         snclId
1         20 2008-06-30 20:00:00 2011-02-18 19:11:00 IU.ANMO.00.BH1
2         20 2008-06-30 20:00:00 2011-02-18 19:11:00 IU.ANMO.00.BH2
3         20 2008-06-30 20:00:00 2011-02-18 19:11:00 IU.ANMO.00.BHZ
4         40 2008-06-30 20:00:00 2011-02-19 06:53:00 IU.ANMO.10.BH1
5         40 2008-06-30 20:00:00 2011-02-19 06:53:00 IU.ANMO.10.BH2
6         40 2008-06-30 20:00:00 2011-02-19 06:53:00 IU.ANMO.10.BHZ
```

The `getAvailability()` method accepts the following arguments:

- `obj` – an `IrisClient` object
- `network` – network code
- `station` – station code
- `location` – location code
- `channel` – channel code
- `starttime` – POSIXct starttime (GMT)
- `endtime` – POSIXct endtime (GMT)
- `includerestricted` – optional flag whether to report on restricted data (default=`FALSE`)

- `latitude` – optional latitude when specifying location and radius
- `longitude` – optional longitude when specifying location and radius
- `minradius` – optional minimum radius when specifying location and radius
- `maxradius` – optional maximum radius when specifying location and radius

3.4 Other IRIS DMC web services

Several methods of the `IrisClient` class work very similarly to the `getAvailability()` method in that they return dataframes of information obtained from web services of the same name. The suite of methods returning dataframes includes:

- `getAvailability`
- `getChannel`
- `getDataselect`
- `getEvalresp`
- `getEvent`
- `getNetwork`
- `getSNCL`
- `getStation`
- `getTraveltime`
- `getUnavailability`

The following example demonstrates the use of several of these services together to do the following:

1. find seismic events on a particular day
2. find available US network BHZ channels in the hour after the biggest event that day
3. determine the easternmost of those channels
4. get the P and S travel times to that station

5. plot the seismic signal detected at that station with markers for P and S arrival times

```
> # Open a connection to IRIS DMC 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 10 degrees of arc of the
> # event location during the hour after the event
> start <- bigOne$time
> end <- start + 3600
> av <- getAvailability(iris, "US", "", "", "BHZ", start, end,
+                       latitude=bigOne$latitude, longitude=bigOne$longitude,
+                       minradius=0, maxradius=10)
> # Get the station the furthest East
> minLonIndex <- which(av$longitude == max(av$longitude))
> snclE <- av[minLonIndex,]
> # Get travel times to this station
> traveltimes <- getTraveltime(iris, bigOne$latitude, bigOne$longitude, bigOne$depth,
+                              snclE$latitude, snclE$longitude)
> # Look at the list
> traveltimes

  distance depth phaseName travelTime rayParam takeoff incident puristDistance
1     8.96  51.9         P     126.67   13.686   86.33    45.55           8.96
2     8.96  51.9         S     226.88   24.532   84.62    47.84           8.96
3     8.96  51.9       PcP     507.26    0.855    3.57     2.56           8.96
4     8.96  51.9       ScS     928.93    1.576    3.67     2.73           8.96
5     8.96  51.9     PKiKP     987.61    0.200    0.84     0.60           8.96
6     8.96  51.9     SKiKS    1406.14    0.224    0.52     0.39           8.96
  puristName
1          P
2          S
3        PcP
4        ScS
5      PKiKP
6      SKiKS

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

```
> sArrival <- start + traveltimes$travelTime[traveltimes$phaseName=="S"]
> # Get the BHZ signal for this station
> st <- getDataselect(iris,snclE$network,snclE$station,
+                     snclE$location,snclE$channel,
+                     start,end)
> # Check that there is only a single trace
> length(st@traces)

[1] 1

> # Plot the seismic trace and mark the "P" and "S" arrival times
> tr <- st@traces[[1]]
> plot(tr, subsampling=1) # need subsmpling=1 to add vertical lines with abline()
> abline(v=pArrival, col='red')
> abline(v=sArrival, col='blue')
```