
]

add SeisIO; build; precompile

test SeisIO

```
using SeisIO
```

```
p = pathof(SeisIO)
d = dirname(realpath(p))
cd(d)
include("../tutorial/install.jl")
```

```
]update
```

```
Ch = SeisChannel()
```

```
?SeisChannel
```

```
Ch = SeisChannel()  
Ch.loc = GeoLoc(lat=-90.0, lon=0.0, el=2835.0, az=0.0, inc=0.0)  
Ch.name = "South pole"
```

```
Ch = SeisChannel(name="Templo de San José de La Floresta, Ajijic", fs=40.0)
```

```
S = SeisData()      # empty structure, no channels  
S1 = SeisData(12)  # empty 12-channel structure
```

```
S = SeisData(1)  
S.name[1] = "South pole"  
S.loc[1] = GeoLoc(lat=-90.0, lon=0.0, el=2835.0, az=0.0, inc=0.0)
```

```
L = GeoLoc(lat=46.1967, lon=-122.1875, el=1440.0, az=0.0, inc=0.0)
S = SeisData(SeisChannel(), SeisChannel())
S = SeisData(randSeisData(5), SeisChannel(),
             SeisChannel(id="UW.SEP..EHZ", name="Darth Exploded", loc=L))
```

```
push!(S, Ch)
```

```
Ch = []
```

```
S = randSeisData(4)
n = 3
append!(S, SeisData(n))
```

```
S = SeisData(2)
S1 = randSeisData(3)
S += S1
```

```
findid(id, S)S.id[i] == idfindchan(cha, S)occursin(cha, S.id[i])
```

```
L = GeoLoc(lat=46.1967, lon=-122.1875, el=1440.0, az=0.0, inc=0.0)
S = SeisData(randSeisData(5), SeisChannel(id="YY.STA1..EHZ"),
             SeisChannel(id="UW.SEP..EHZ", name="Darth Exploded", loc=L))
findid("UW.SEP..EHZ", S)      # 7
findchan("EHZ", S)           # [6, 7], maybe others (depending on randSeisData)
```

```
deleteat!(S, 1:2)  # Delete first two channels of S
S -= 3             # Delete third channel of S
```

```
# Extract S[1] as a SeisChannel, removing it from S  
C = pull(S, 1)
```

```
# Delete channels containing ".SEP."  
delete!(S, ".SEP.", exact=false)
```

```
# Delete all channels whose S.x is empty  
prune!(S)  
S
```

```
delete!(S, str)
```

findchan()

findchan()

€

findid()

findid()

namestrip!(S[, convention])

	"\$*/:<>?@\^ ~DEL
	"&' ;<>@DEL
	\$/DEL
	!#()*+-. [\]_`{ }
	.DEL
	!"#\$%&'()*+,-./:;<=>?@[\]^`{ }~DEL

DEL

timestamp()

track_off!(S)

track_on!(S)

prune!(S::SeisData)

C = pull(S::SeisData, id::String)

C = pull(S::SeisData, i::integer)

note!(S, i, str)

note!(S, id, str)

note!(S, str)

iS

clear_notes!(S, id)

Sid

clear_notes!(S)

S

show_processing()

show_processing()

show_processing()

show_src()

show_src()

show_src()

show_writes()

show_writes()

show_writes()

```
p = pathof(SeisIO)
d = dirname(realpath(p))
cd(d)
include("../test/examples.jl")
```

```
using Pkg
Pkg.test("SeisIO")      # lunch break recommended. Tests can take 20 minutes.
                        # 99.5% code coverage wasn't an accident...
p = pathof(SeisIO)
cd(realpath(dirname(p) * "../test/"))
```

SeisData

A custom structure designed to contain the minimum necessary information for processing univariate geophysical data.

SeisChannel

A single channel designed to contain the minimum necessary information for processing univariate geophysical data.

Fields

=====

Field	Description
-------	-------------

:n	Number of channels [¹]
:c	TCP connections feeding data to this object [¹]
:id	Channel id. Uses NET.STA.LOC.CHAN format when possible
:name	Freeform channel name
:loc	Location (position) vector; any subtype of InstrumentPosition
:fs	Sampling frequency in Hz; fs=0.0 for irregularly-sampled data.
:gain	Scalar gain
:resp	Instrument response; any subtype of InstrumentResponse
:units	String describing data units. UCUM standards are assumed.
:src	Freeform string describing data source.
:misc	Dictionary for non-critical information.
:notes	Timestamped notes; includes automatically-logged info.
:t	Matrix of time gaps in integer μ s, formatted [Sample# Length]
:x	Time-series data

dataless_support()

mseed_support()

resp_wont_read()

seed_support()

suds_support()

?web_chanspec

?seis_www

?TimeSpec

```
read_data!(S, fmt::String, filepath [, Kws])  
S = read_data(fmt::String, filepath [, Kws])
```

filepath

?SeisIO.KW

```
SeisIO.KW.nx_new = 60000  
SeisIO.KW.nx_add = 360000
```

```
strict=true
```

```
S = read_data("uw", "99011116541W", full=true)
```

```
99011116541W
```

```
:misc
```

```
read_data!(S, "sac", "MSH80*.SAC")
```

```
S
```

```
S = read_data("win32", "20140927*.cnt", cf="20140927*ch", nx_new=360000)
```

```
2014092709*.cnt
```

```
20140927*ch
```

```
nx_new
```

```
SeisIO.formats("FMT")keys(SeisIO.formats)
```

rseis()

sachdr()

segyszer([])

passcal=true

```
read_meta!(S, fmt::String, filepath [, Kws])  
S = read_meta(fmt::String, filepath [, Kws])
```

filepath

?SeisIO.KW

```
S = read_hdf5(fname::String, s::TimeSpec, t::TimeSpec, [, Kws])
read_hdf5!(S::GphysData, fname::String, s::TimeSpec, t::TimeSpec, [, Kws])
```

?SeisIO.KW

`read_sxml([])`

s
t
msr

`read_qml()`

get_data!

get_data!(S, method, channels; Kws)
S = get_data(method, channels; Kws)

src?seis_www

?chanspec

unscale
demean
detrrend
taper
ungap
rr

autoname
msrMultiStageResonse
s
t
xf

autoname=true

FDSNsta!(S, chans, KW)
S = FDSNsta(chans, KW)

msrMultiStageResonse

s
t
xf

```
S = get_data("FDSN", "CC.VALT, UW.SEP, UW.SHW, UW.HSR", src="IRIS", t=-600)
S -= "UW.SHW..ELZ"
S -= "UW.HSR..ELZ"
writesac(S)
```

```
CHA = "CC.PALM, UW.HOOD, UW.TIMB, CC.HIYU, UW.TDH"
TS = u2d(time())
TT = -600
S = get_data("FDSN", CHA, src="IRIS", s=TS, t=TT)
```

```
ts = "2011-03-11T06:00:00"
te = "2011-03-11T06:05:00"
R = get_data("FDSN", "GE.BKB..BH?", src="GFZ", s=ts, t=te, v=1, y=true)
```

```
S = FDSNsta("CC.VALT.., PB.B001..BS?, PB.B001..E??")
```

TSTTCHA

```
S = SeisData()
CHA = "UW.TDH..EHZ, UW.VLL..EHZ, CC.VALT..BHZ"
TS = u2d(time()-86400)
TT = 600
get_data!(S, "IRIS", CHA, s=TS, t=TT)
```

```
CHA = "UW.TDH..EHZ, UW.VLL..EHZ, CC.VALT..BHZ"
TS = u2d(time())
TT = -600
S = get_data("IRIS", CHA, s=TS, t=TT, y=true, to=55, w=true)
```

```
STA = "UW.HOOD.--.BHZ,CC.TIMB.--.EHZ"  
TS = "2016-05-16T14:50:00"; TE = 600  
S = get_data("IRIS", STA, s=TS, t=TE)
```

```
ts = "2016-03-23T23:10:00"  
te = "2016-03-23T23:17:00"  
S = get_data("IRIS", "CC.JRO..BHZ", s=ts, t=te, fmt="sacbl")  
T = get_data("IRIS", "CC.JRO..BHZ", s=ts, t=te, fmt="miniseed")
```

```
:misc["msg"]println(stdout, S.misc[i]["msg"])
```

```
:misc["raw"]
```

```
seedlink!(S, mode, chans, Kws)
seedlink!(S, mode, chans, patts, Kws)
S = seedlink(mode, chans, Kws)
chanspattsSS.c
```

			1

SeedLink follows unusual rules for wild cards in **sta** and **patts**:

*

sta

DO NOT feed one data channel from multiple SeedLink connections. This leads to TCP congestion on your computer, v

`kill -9`

`close(S.c[i])i`

`!deleteat(S.c, i)i`

SL_info()

has_sta([])

¹ This value is a base value; a small amount is added to this number by each new SeedLink session to minimize the risk of congestion

u

has_stream()

has_stream()

u

`write_hdf5()`

`write_qml()`

`write_qml()`

`write_qml()`

writesac()

writesac()

writesacpz([])

wseis()

wseis()

write_sxml([])

nanfill!(S)

resample!(S::GphysData [, chans=CC, fs=FS])

resample!(C::SeisChannel, fs::Float64)

unscale!(S[, chans=CC, irr=false])

```
convert_seis!(S[, chans=CC, units_out=UU, v=V])  
convert_seis!(C[, units_out=UU, v=V])
```

```
ungap!(S[, chans=CC, m=true, tap=false])  
ungap!(C[, m=true, tap=false])
```

```
merge!(S::GphysData, U::GphysData)
```

```
merge!(S::GphysData)
```

:id:fs:loc:resp:units
:loc:resp:units

merge!

:id:fs:loc:resp:units

mseis!(S::GphysData, U::GphysData, ...)

translate_resp!(S, resp_new[, chans=CC, wl=g])
translate_resp!(Ch, resp_new[, wl=g])

remove_resp!(S, chans=CC, wl=g)

remove_resp!(Ch, w1=g]

detrend!taper!

sync!(S::GphysData[, s=ST, t=EN, v=VV])

SeisIO.KW.Filt.np = 2

eps(Float32)

using SeisIO.Quake

randPhaseCat()

randSeisChannel([])

randSeisData([])

randSeisData([])

randSeisEvent([])

randSeisEvent([])

randSeisHdr()

randSeisSrc()

dataless_support()

mseed_support()

seed_support()

scan_seed()

`get_data(..., w=true)`

`uwpf([])`

`uwpf!(W, pf[, v::Int64=KW.v])`

S = read_nodal(fmt, fname [, Kws])

s=t=now()

-
-
-

-
-
- :resp

S.t[i]
S.fs[i]
length(S.x[i])

:datahcat()

:t:fs:data

:fs = 0.0

NodalLoc()

NodalData()

NodalChannel()

EQMag()

EQLoc()

EventChannel()

EventTraceData()

PhaseCat()

SeisEvent()

SeisHdr()

SeisPha()

SeisSrc()

SourceTime()

FDSNeqv()

src="IRIS, INGV, NCEDC"

FDSNevt()

get_pha!(S::Data[, keywords])

		1	
			2
			3
			4
			5
			6
			7
			8

```
S = FDSNevt("201103110547", "PB.B004..EH?,PB.B004..BS?,PB.B001..BS?,PB.B001..EH?")
wseis("201103110547_evt.seis", S)
```

distaz!(Ev::SeisEvent)

gcdist([])

show_phases()

fill_sac_evh!(Ev::SeisEvent, fname[; k=N])

¹ Types: A = Array, B = Boolean, C = Char, DT = DateTime, F = Float, I = Integer, S = String, U8 = Unsigned 8-bit integer (UInt8)

² search range is always $ot - |evw[1]| \leq t \leq ot + |evw[2]|$

³ nev=0 returns all events in the query

⁴ String is passed as-is, e.g. szsrecs=true&repo=realtime for FDSN. String should not begin with an ampersand.

⁵ Comma-separated String, like P, pP; use ttall for all phases

⁶ Specify region [center_lat, center_lon, min_radius, max_radius, dep_min, dep_max], with lat, lon, and radius in decimal degrees (ř) and depth in km with + = down. Depths are only used for earthquake searches.

⁷ Specify region [lat_min, lat_max, lon_min, lon_max, dep_min, dep_max], with lat, lon in decimal degrees (ř) and depth in km with + = down. Depths are only used for earthquake searches.

⁸ If w=true, a file name is automatically generated from the request parameters, in addition to parsing data to a SeisData structure. Files are created from the raw download even if data processing fails, in contrast to get_data(wsac=true).

S = read_quake(fmt::String, filename [, Kws])

?SeisIO.KW

read_qml()

`write_qml()`

asdf_waux()

asdf_rqml()

asdf_wqml([])

asdf_wqml([])

read_asdf_evt([])

read_asdf_evt([])

scan_hdf5()

scan_hdf5()

YYYY-MM-DDThh:mm:ss.ssssss

- μ

-

-

NN.SSSSS.LL.CC

NN.SSSSS.LL.CC.TTDECOTL

chanspec()

?chanspec

"UW,ELK,.EHZ"["UW" "ELK" "" "EHZ"]--*??

?CC.VALT..???

*CC.VALT..*CC.VALT..???

"UW,EL?"" .ELK.."
