MESS 2011 – ObsPy Introduction

Robert Barsch, Tobias Megies

Department für Geo- and Umweltwissenschaften (Geophysik)
Ludwig-Maximilians-Universität München

2011-02-22

ObsPy Data Types – Why bother?

- ... we want to unify data from different sources in a common structure.

```
st = read("file.mseed")
st += read("file.sac")
st += client_arclink.getWaveform(...)
st += client_iris.getWaveform(...)
```

LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

ObsPy Data Types – Why bother?

- ... they know how to behave by themselves if we tell them once.

```
utcdatetime + 10
st += st2
st.filter("lowpass", freq=1)
```

ObsPy Data Types – Why bother?

- ... there is less room for user errors.

```
#st = client.getWaveform(..., channel="BHZ")
st = client.getWaveform(..., channel="HHZ")
data = st[0].data

data = obspy.signal.lowpass(data, freq=1, df=20)
```

ObsPy Data Types – Why bother?

- ... the code gets much shorter and better readable.

How about...

```
st = read("file")
from obspy.signal import lowpass
num_traces = len(st)
for i in range(num_traces):
    df = st[i].stats.sampling_rate
    st[i].data = lowpass(st[i].data, freq=1, df=df)
```

...against:

```
st = read("file")
st.filter("lowpass", freq=1)
```

ObsPy Data Types – Overview

- `UTCDateTime`
  - extension of the Python `datetime` object
  - stores a time stamp
- `Stats`
  - extension of the Python `dict` object
  - stores header information of waveforms
- `Trace`
  - stores a single-channel, continuous piece of waveform data
  - consisting of waveform data and header information
- `Stream`
  - stores multiple traces (e.g. Z, N, E traces of one station)
- all of them defined in `obspy.core`

ObsPy Data Types – `UTCDateTime`

- `UTCDateTime`
  - used to handle all time information in ObsPy
  - initialize via
    - `t = UTCDateTime("2011-02-21T08:00:00.00Z")`
    - `t = UTCDateTime(2011, 2, 21, 8)`
    - ...
  - several attributes/methods
    (e.g. `t.microsecond`, `t.julday`, `t.weekday()`, ...)
  - important operations
    - subtracting two `UTCDateTime` objects gives time difference in seconds
    - adding/subtracting `int/float` returns new `UTCDateTime` object
  - see ObsPy documentation

ObsPy Data Types – `Stats`

- `Stats` – header information for waveform data
  - contains at least the following keys
    - `stats.network` – network code (`str`)
    - `stats.station` – station code (`str`)
    - `stats.location` – location code (`str`)
    - `stats.channel` – channel code (`str`)
    - `stats.starttime` – time of first sample (`UTCDateTime`)
    - `stats.sampling_rate` – sampling rate in `Hz` (`float`)
    - `stats.npts` – number of samples (`int`)
  - derived keys
    - `stats.endtime` – time of last sample (`UTCDateTime`)
    - `stats.delta` – time interval between two samples (`float`)
  - optional keys
    - `stats._format` – format of original data file (`str`, e.g. `"MSEED"`)
    - `stats.paz` – poles, zeros, sensitivity and gain of instrument (`dict`)
    - `stats.coordinates` – longitude, latitude and elevation of station (`dict`)
    - ...
  - see ObsPy documentation

ObsPy Data Types – `Trace`

- `Trace` – continuous waveform data
  - usually constructed internally during `read(...)` or `getWaveform(...)`
  - consists of
    - `tr.data` – waveform data as a `numpy.ndarray` instance
    - `tr.stats` – header information as a `Stats` instance
  - built-in methods
    - `tr.id` – complete channel id in SEED standard (e.g. `"BW.RJOB..BHZ"`)
    - `tr.plot()` – shows preview plot of trace
    - `tr.copy()` – returns copy of trace (most operations work in-place)
    - `tr.trim(starttime, endtime)` – cut trace to specified time span
    - `tr.filter("type", **kwargs)` – filter waveform data
    - `tr.simulate(paz_remove, paz_simulate, **kwargs)` – apply instrument correction/simulation
    - `tr.write("filename", "format")` – write waveform to local file
    - ...
  - many built-in methods on `tr.data` (`numpy.ndarray`)!
  - see ObsPy documentation
  - see Numpy documentation – ndarray

ObsPy Data Types – `Stream`

- `Stream` – collection of `Trace` objects in a `list`-like container
  - usually returned by a `read(...)` or `getWaveform(...)` call
  - `print st` – prints summary of all traces
  - `print len(st)` – prints number of traces in stream
  - `list`-like operations
    - `st[i]` – return trace at index `i`
    - `st.append(tr)` – add a single trace
    - `st.extend(st)` – add a list of traces
    - `st.remove(tr)` – remove specified trace from stream
    - `st.pop(i)` – remove trace at specified index and return it
    - `st.sort(...)` – sort traces in stream according to specified criteria
  - other built-in methods
    - `st.select(**kwargs)`
      – return new stream with matching traces (e.g. `component="Z"`)
    - `st.merge(method)` – merge traces with identical id
    - `st.printGaps()` – prints summary of gaps in the stream
  - many built-in methods of `Trace` (`trim`, `filter`, `simulate`,...)
  - see ObsPy documentation

Getting Help..

IPython

- get help for a function: `>>> command?`
- have a look at the implementation: `>>> command??`
- search for variables/functions/modules starting with "ab": `>>> ab<Tab>`
- what's the value? `>>> variable`
- what's the type? `>>> type(variable)`
- which variables are assigned anyway?? `>>> whos`
- what attributes/methods are there? `>>> variable.<Tab>`
- get help for a variable's method: `>>> variable.command?`
- what functions are available in a module? `>>> module.<Tab>`

Getting Help..

- ObsPy web pages
  - Tutorial
    - http://obspy.org/wiki/ObspyTutorial
    - file:///home/messuser/obspy/tutorial/ObspyTutorial.html
  - API
    - http://docs.obspy.org/
    - file:///home/messuser/obspy/docs/index.html
- Python/Numpy/Scipy API
  - http://docs.python.org/
  - file:///home/messuser/obspy/python/python-docs/index.html
  - http://docs.scipy.org/doc/numpy/reference/
  - file:///home/messuser/obspy/python/numpy-docs/index.html
  - http://docs.scipy.org/doc/scipy/reference/
  - file:///home/messuser/obspy/python/scipy-docs/index.html

How to Work on the Practicals..

- Either..
  - work line by line in IPython shell
  - when it's working: save history and condense it

```
>>> %history [number_of_lines] [-n] [-f output_file]
```

- or..
  - work on your program in a text editor
  - in a second window, run program in an IPython shell and continue work at the end

```
$ ipython -i
>>> run -i PROGRAM.PY
```

    - (caution: best do this in a "fresh" IPython shell)
  - extend program with appropriate lines of code and run it again in a new IPython shell