ObsPy: A Python Toolbox for Seismology

Last Updated: April 2013

Lion Krischer

Ludwig-Maximilians-University in Munich
Department of Earth and Environmental Sciences
Geophysics

1. Python

2. ObsPy

Why use Python?

- Easy
  $\Rightarrow$ Learning curve similar to Matlab
- Free
- Runs everywhere
- Code should be a tool and work for you - not against you

Why use Python?

- Open Source
- Cross-platform
- General purpose language (in contrast to other tools used in science)
- No need to compile
- Interactive shell
- Readability
- Speed
- “Batteries included”
- Mature third party libraries
- Makes it easy to interact with existing C and Fortran code

Why use Python?

- One of the most used programming languages
- Huge community outside of science
  $\Rightarrow$ Tools and support widely available
- Used a lot in the web community
  $\Rightarrow$ Will become more and more important
- Big support from the data analysis community
- Interesting new developments: pypy, blaze, numba, ipython, ...
- Future proof

Must have third party libraries/tools

- **NumPy/SciPy** - array and matrix structures, linear algebra routines, numerical optimization, random number generation, statistics routines, differential equation modeling, Fourier transforms and signal processing, image processing, sparse and masked arrays, spatial computation, and numerous other mathematical routines
- **matplotlib** - 2D (and limited 3D) plotting library which produces publication quality figures via a set of functions familiar to MATLAB users
- **basemap** - Maps
- **IPython** - Enhanced interactive shell, also provides a HTML notebook
- **lxml**, **PyQt**, **Pandas**, **pytables**, **SQLAlchemy** . . .

Readable Syntax

```
>>> 3 in [1, 2, 3, 4, 5]
True
>>> "sub" in "substring"
True


>>> open_file = open("filename", "r")
>>> for line in open_file:
...     print line


>>> for i in [1, 2, 3, 4, 5]:
...   if i < 3:
...       print i,
...   else:
...       print i * 2,
1 2 6 8 10
```

"Batteries included"

- Extensive standard libraries:
  - **Powerful string handling**
  - **Filesystem manipulation utitilties**
  - Data Persistence
  - Data Compression and Archiving
  - Cryptographic Services
  - **Internet Protocols and Data Handling**
  - Structured Markup Processing Tools
  - Development Tools
  - Multithreading & Multiprocessing
  - **Regular expressions**
  - Graphical User Interfaces with Tk
  - ...

”Batteries included”

- Well-documented
- Platform independent API, but optimized for each platform
- One place to look first for a proven solution
- Reuse instead of reinvent

Speed

“Python is extremely slow and wastes memory!”

Implementation Time vs. Execution Time

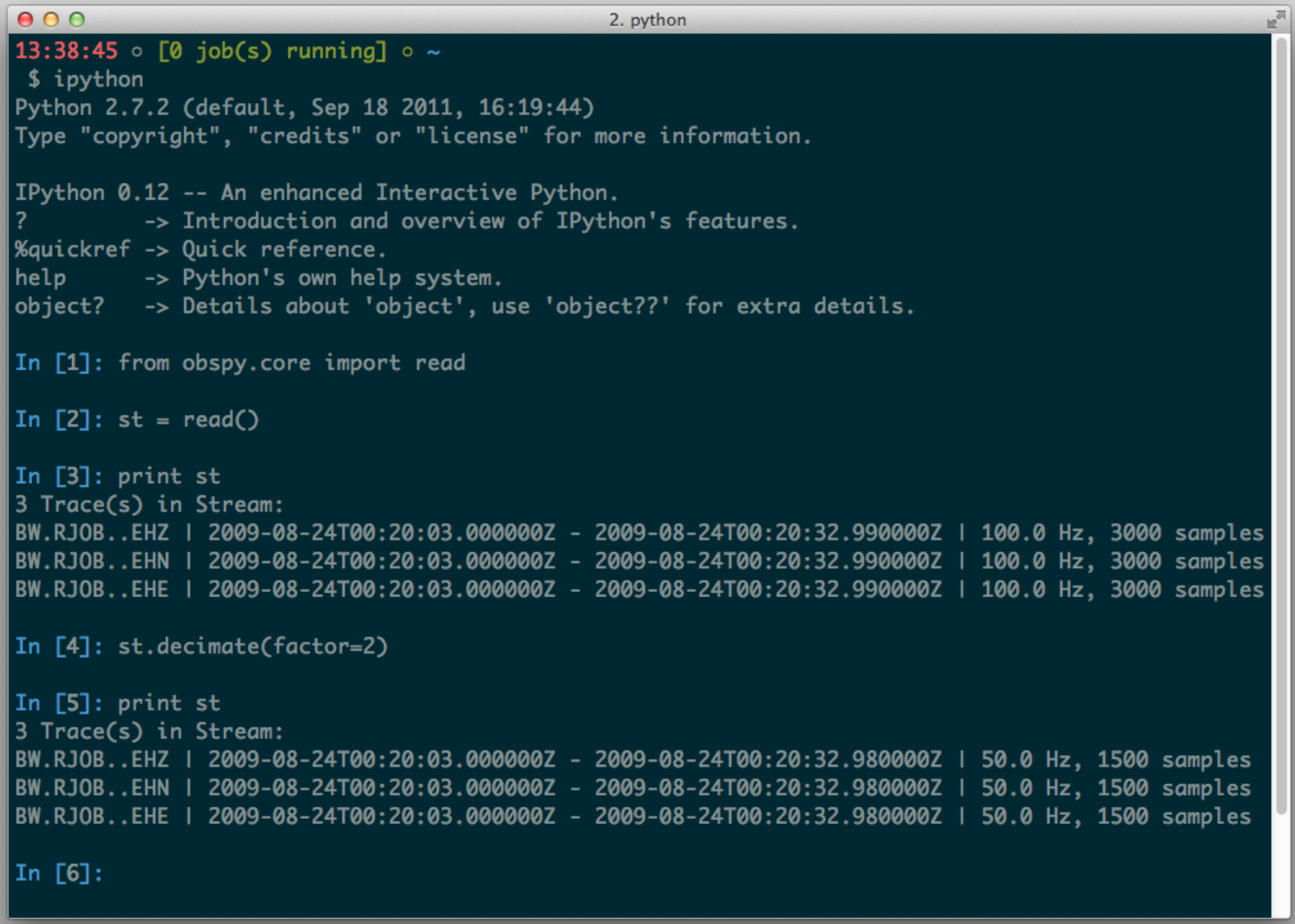
- Python is designed for productivity
- No compilation step
  - No compiler problems
  - No makefiles
  - No linker problems
  - Faster development cycles
- When execution speed matters:
  - Use specialized modules
  - Implement time critical parts in C/C++/Fortran/Cython
  - Profile
  - Use specialized JIT compiler (e.g. PyPy or Numba)
- Python is fast enough in most cases.
- Otherwise choose the right tool!

Language Interoperability

Python excels at gluing other languages together:

- **FORTRAN**: F2py - Fortran to Python interface generator (part of NumPy).
- General **C** or **C++** libraries: Ctypes, Cython, or SWIG are three ways to interface them.
- **R**: RPy - simple, robust Python interface to the R Programming Language. It can manage all kinds of R objects and can execute arbitrary R functions (including the graphic functions).
- **Matlab**: The similarity of Matlab and NumPy/SciPy/matplotlib makes it very easy to transfer existing code. NumPy can directly read the files saved by Matlab.

IPython Console



IPython HTML Notebook

Untitled4

http://127.0.0.1:8888/0fadbca9-66c7-4cd6-a943-5a984c840784

Google

IP[y]: Notebook

Example | Rename | QuickHelp

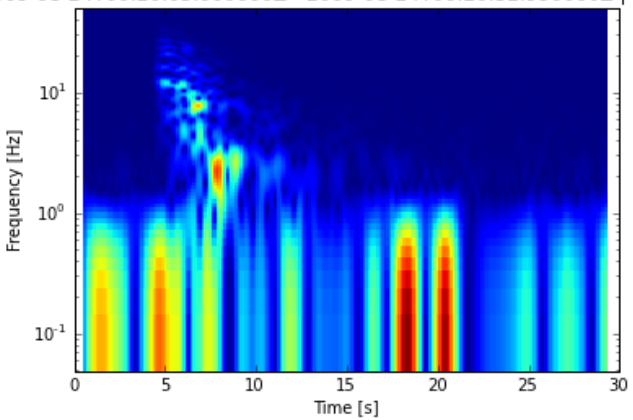
In [1]:
```
from obspy.core import read
st = read()[0]
print st
```

```
BW.RJOB..EHZ | 2009-08-24T00:20:03.000000Z - 2009-08-24T00:20:32.990000Z | 100.0 Hz, 3000 samples
```

In [2]:
```
st.plot()
```

In [3]:
```
st.spectrogram(log=True)
```

BW.RJOB..EHZ | 2009-08-24T00:20:03.000000Z - 2009-08-24T00:20:32.990000Z | 100.0 Hz, 3000 samples



ObsPy

What is ObsPy and what can it do

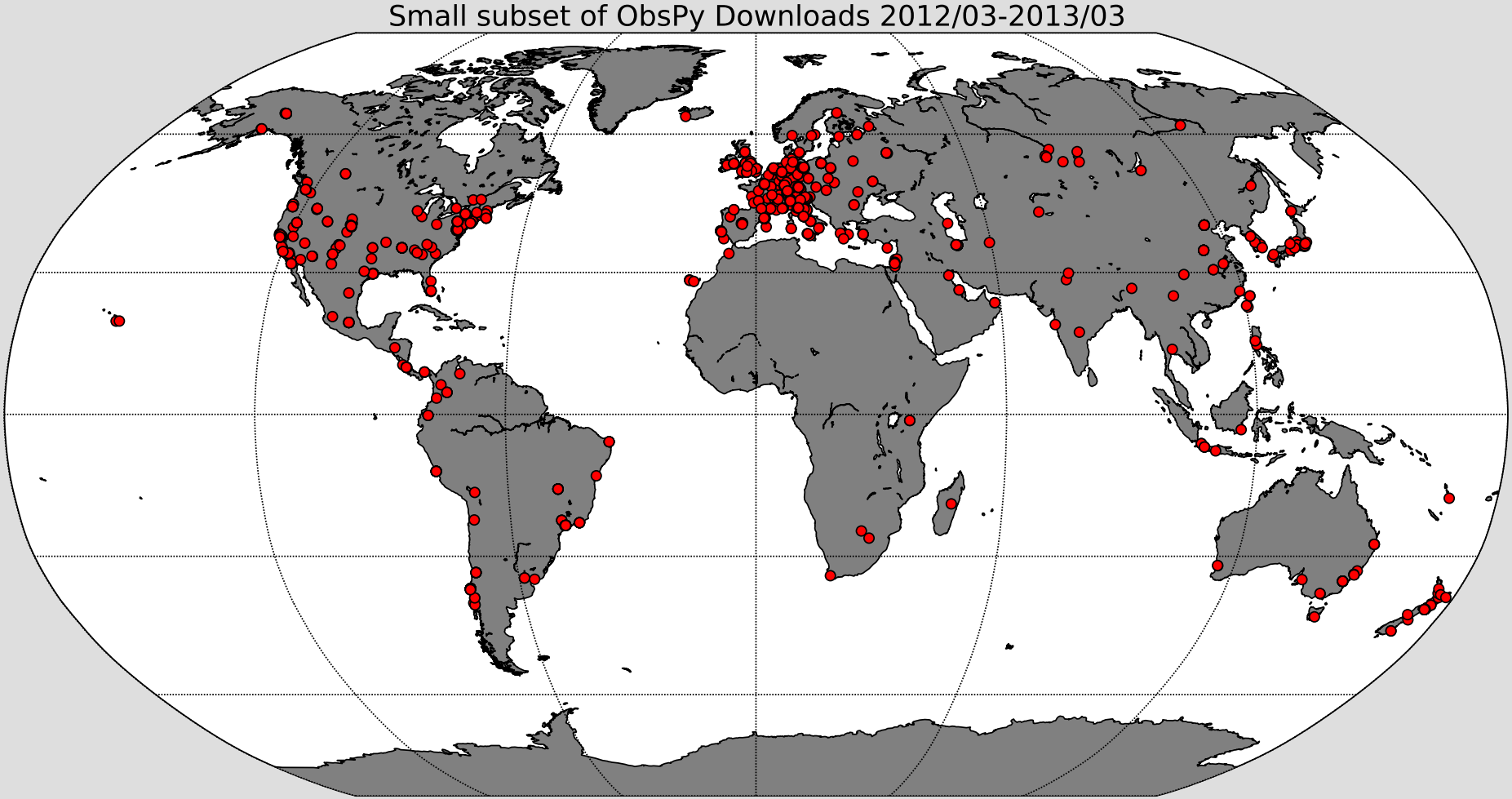
A Python Toolbox for seismology/seismological observatories.

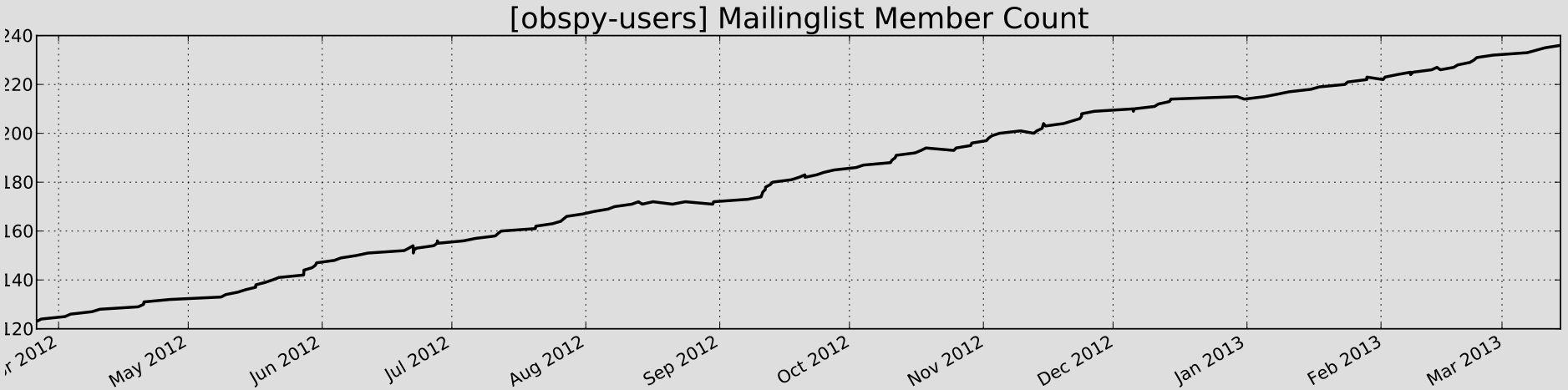
The goal of the ObsPy project is to **facilitate rapid application development for seismology**.

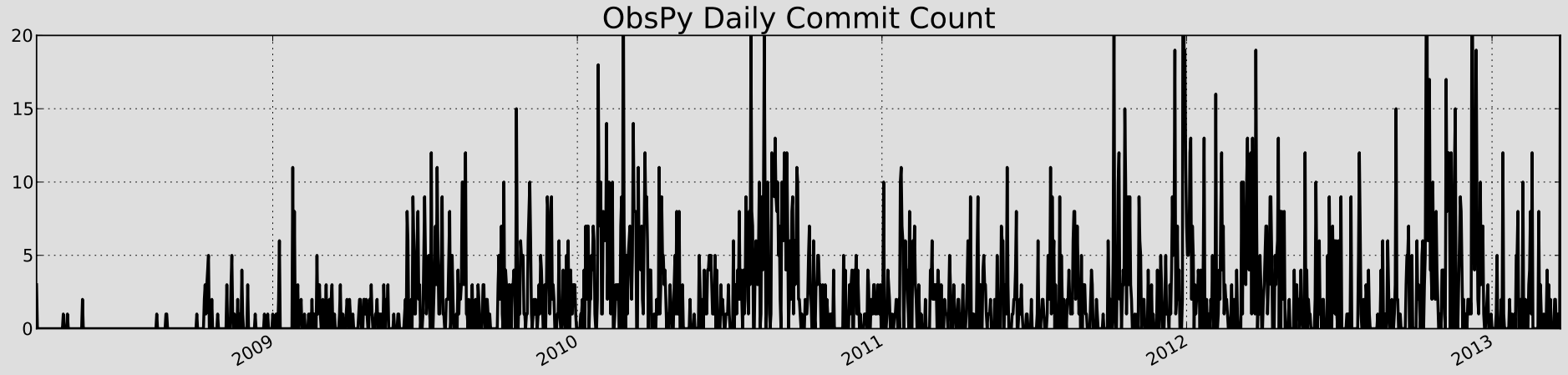
http://www.obspy.org

What is ObsPy and what can it do

- In development since 2008
- 6 core developers
- Many more people contribute (currently around 25 people contributed code)
- Thoroughly unit tested
- Written in Python (performance critical parts are written in C)
- Uses well established libraries (libmseed, GSE_UTI, . . .)
- Open source and cross platform
- Starts to get widely used in the community
  $\Rightarrow$ Estimated active user base: $>$ 5000


Small subset of ObsPy Downloads 2012/03-2013/03


[obspy-users] Mailinglist Member Count
240
220
200
180
160
140
120
r 2012
May 2012
Jun 2012
Jul 2012
Aug 2012
Sep 2012
Oct 2012
Nov 2012
Dec 2012
Jan 2013
Feb 2013
Mar 2013


ObsPy Daily Commit Count
20
15
10
5
0
2009
2010
2011
2012
2013

What is ObsPy and what can it do

- **Read and write waveform data in various formats** (MiniSEED, SAC, GSE, SEG Y, ...) with a unified interface.
- **Database and webservice access clients** for NERIES, IRIS DMC, ArcLink, SeisHub and Earthworm (experimental).
- **Many seismological signal processing routines** like filters, trigger, instrument correction, array analysis, beamforming, ...
- **Support for inventory data** (SEED, XSEED, RESP and planned StationXML support)
- **Event data handling**

+

The full power and flexibility of Python.

Simple example - Reading a waveform file

```
>>> from obspy import read
>>> st = read("waveform.mseed")
>>> print st
1 Trace(s) in Stream:
BW.FURT..EHZ | 2010-01-04 ... | 200.0 Hz, 7204234 samples
```

- Automatic file format detection.
- Always results in a Stream object.
- Raw data available as a numpy.ndarray.

```
>>> st[0].data
array([-426, -400, ... , -489, -339], dtype=int32)
```

Working with the waveform data.

```
>>> print st
1 Trace(s) in Stream:
BW.FURT..EHZ | 2010-01-05T ... | 200.0 Hz, 17280068 samples
>>> st.trim(endtime=st[0].stats.starttime + 3600)
>>> st.decimate(factor=2)
>>> st.filter("lowpass", freq=1.0)
>>> print st
BW.FURT..EHZ | 2010-01-05T ... | 100.0 Hz, 360001 samples
>>> st.plot()
```

The resulting Stream object

- A **Stream** object is a collection of **Trace** objects
- A **Trace** object is a single, continuous waveform data block
- Working with them is easy:
  - **st.filter()** - Filter all attached traces.
  - **st.trim()** - Cut all traces.
  - **st.resample() / st.decimate()** - Change the sampling rate.
  - **st.trigger()** - Run triggering algorithms.
  - **st.plot() / st.spectrogram()** - Visualize the data.
  - **st.simulate(), st.merge(), st.normalize(), st.detrend(), ...**
- A **Stream** object can also be exported to many formats making ObsPy a good tool for converting between different file formats.

```
>>> st.write("output_file.sac", format="SAC")
```

Handling time - The UTCDateTime class

- All absolute time values are consistently handled with this class.
- No ambiguities, e.g. timezones, leap seconds, ...
- Based on a high precision POSIX timestamp
- Easy usage:

```
>>> from obspy import UTCDateTime
>>> time = UTCDateTime(2012, 9, 12, 8, 0, 0)
>>> one_hour_later = time + 3600

>>> UTCDateTime() - time
8078.097172
```

Inventory Data

- Can currently read/write/convert between SEED and XML-SEED.
- RESP file support.
- StationXML support is planned.

```
000001V 010009402.3121970,001,00:00:00.0000~2038,001,00:00:00.0000~
2009,037,04:32:41.0000~BayernNetz~~0110032002RJOB 000003RJOB 000008
...
```

```
<?xml version='1.0' encoding='utf-8'?>
<xseed version="1.0">
  <volume_index_control_header>
    <volume_identifier blockette="010">
      <version_of_format>2.4</version_of_format>
      <logical_record_length>12</logical_record_length>
      <beginning_time>1970-01-01T00:00:00</beginning_time>
      <end_time>2038-01-01T00:00:00</end_time>
...
```

Inventory Data - Example usage

```
>>> from obspy.xseed import Parser
>>> p = Parser("dataless_SEED")
>>> print p
BW.FURT..EHZ | 2001-01-01T00:00:00.000000Z -
BW.FURT..EHN | 2001-01-01T00:00:00.000000Z -
BW.FURT..EHE | 2001-01-01T00:00:00.000000Z -
>>> p.getCoordinates("BW.FURT..EHZ")
{"elevation": 565.0, "latitude": 48.162899,
 "longitude": 11.2752}
>>> p.getPAZ("BW.FURT..EHZ")
{"digitizer_gain": 1677850.0,
 "gain": 1.0,
 "poles": [(-4.444+4.444j), (-4.444-4.444j), (-1.083+0j)],
 "seismometer_gain": 400.0,
 "sensitivity": 671140000.0,
 "zeros": [0j, 0j, 0j]}
```

Events

- Aims to get a unified interface with read and write support independent of the data source, similar to how the Stream and Trace classes handle waveform data.
- Modelled after QuakeML - currently supports QuakeML 1.2

```
>>> from obspy import readEvents
>>> url = "http://www.seismicportal.eu/services/..."
>>> catalog = readEvents(url)
>>> print catalog
99 Event(s) in Catalog:
2012-04-11T10:43:09.400000Z |  ... | 8.2 Mw | ...
2012-04-11T08:38:33.000000Z |  ... | 8.4 M  | ...
...
```

The Catalog object

- The Catalog object contains some convenience methods to make working with events easier.
- Events can be filtered with various keys.

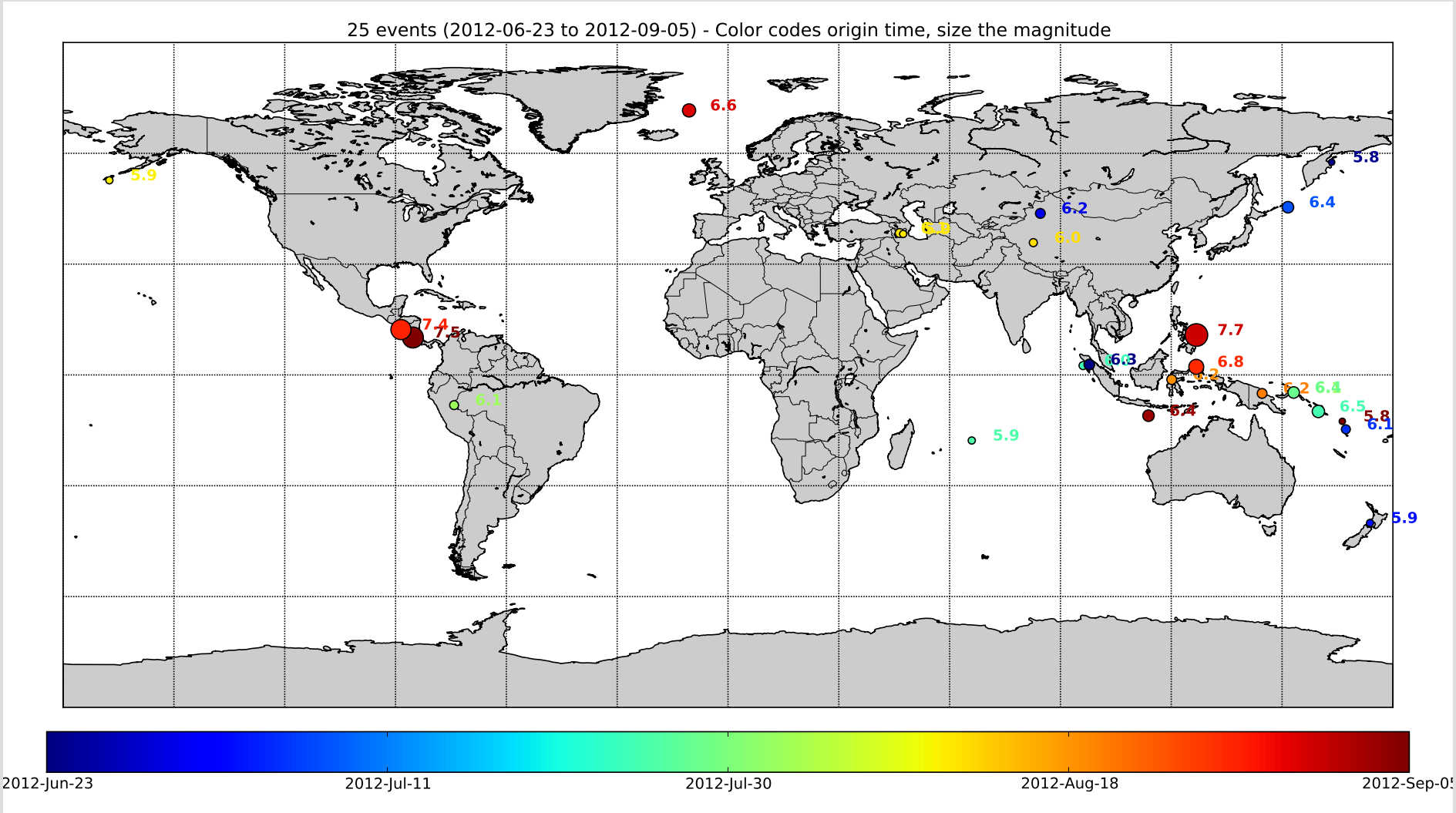
```
>>> small_magnitude_events = cat.filter("magnitude <= 4.0")
```

- They can be plotted using the basemap module.

```
>>> cat.plot()
```

- And they can be written.

```
>>> cat.write("modified_events.xml", format="quakeml")
```


25 events (2012-06-23 to 2012-09-05) - Color codes origin time, size the magnitude
6.6
5.9
5.8
6.4
6.2
6.0
7.4
7.5
7.7
6.8
6.1
5.9
5.9
2012-Jun-23
2012-Jul-11
2012-Jul-30
2012-Aug-18
2012-Sep-05

Clients - Getting waveform data from the web

Available for **NERIES**, **IRIS**, **ArcLink**, **SeisHub** and **Earthworm**.

```
>>> from obspy.neries import Client
>>> from obspy import UTCDateTime
>>> client = Client(user="test@obspy.org")
>>> dt = UTCDateTime("2009-08-20 04:03:12")
>>> st = client.getWaveform("BW", "RJOB", "", "EH*", \
                            dt - 3, dt + 15)
>>> print st
3 Trace(s) in Stream:
BW.RJOB..EHN | 2009-08-20T04: ... | 200.0 Hz, 3601 samples
BW.RJOB..EHE | 2009-08-20T04: ... | 200.0 Hz, 3601 samples
BW.RJOB..EHZ | 2009-08-20T04: ... | 200.0 Hz, 3601 samples
```

- Similar interfaces for the other clients.
- The returned object is the same as if read from a file.

Clients - Retrieving other data

The webservices are not limited to retrieving waveform data. Depending on the client module used, the available data includes:

- Events information
- Inventory and response data
- Availability information
- ...

obspy.signal - Signal Processing Routines

sonic
array_transff_wavenumber
array_transff_freqslowness
relcalstack
envelope
normEnvelope
centroid
instFreq
instBwith
xcorr
xcorr_3C
xcorr_max
xcorrPickCorrection
simple
bandpass
bandstop
lowpass
highpass
envelope
remezFIR
lowpassFIR
integerDecimation
lowpassCheby2
polarizationFilter
cfrequency
bwith
domperiod
logbankm
logcep
sonogram
cosTaper
c_sac_taper
evalresp
cornFreq2Paz
pazToFreqResp
waterlevel
specInv
seisSim
paz2AmpValueOfFreqResp
estimateMagnitude
estimateWoodAndersonA...
konnoOhmachiSmoothing
eigval
class PPSD
rotate_NE_RT
rotate_ZNE_LQT
rotate_LQT_ZNE
cwt
fem
fpm
em
pm
tpg
rdct
fpg
eg
pg
plotTfMisfits
plotTfGofs
plotTfr
recSTALTA
carlSTATrig
classicSTALTA
delayedSTALTA
zDetect
triggerOnset
pkBaer
arPick
plotTrigger
coincidenceTrigger
utlGeoKm
utlLonLat

Complete Workflow Example

```
from obpsy import UTCDateTime
from obspy.arclink import Client()
client = Client()
t0 = UTCDateTime("2012-09-08 20:30:00")
# Download data and instrument response.
st = client.getWaveform ("NA", "SABA", "", "BHZ", \
    t0 - 900, t0 + 900)
paz = client.getPAZ ("NA", "SABA", "", "BHZ", t0 )
# Add a zero to convert to m (default is m/s)
paz["zeros"].append(0 + 0j)
st.simulate(paz_remove=paz)
# Apply highpass filters
st.filter("highpass",freq=0.005)
# Trim the signal to your needs
st.trim(t0 - 60, t0 + 60)
# Save and plot the deconvolved data
st.write("deconvolved_data", format="mseed")
st.plot()
```

What's next?

- **Getting more developers and external contributions**
  $\Rightarrow$ We try to be as open as possible and very much encourage people to add to/modify our code basis
- Adding support for Python 3 (partially done)
- Support for StationXML (partially done)
- More powerful instrument correction module
- Suggestions? Let us know!

www.obspy.org

- Documentation and extensive tutorial.
- Gallery to showcase some features.
- **mailing list** - subscribe for updates and discussions about the project.
- Source code repository and bug tracker.
- Automatic, daily running tests bots (http://tests.obspy.org)
- Get in touch!

- Moritz Beyreuther et al. (2010) **ObsPy: A Python Toolbox for Seismology**, SRL, 81(3), 530-533
- Tobias Megies et al. (2011) **ObsPy – What can it do for data centers and observatories?** Annals Of Geophysics, 54(1), 47-58. doi:10.4401/ag-4838

This Page

Show Source

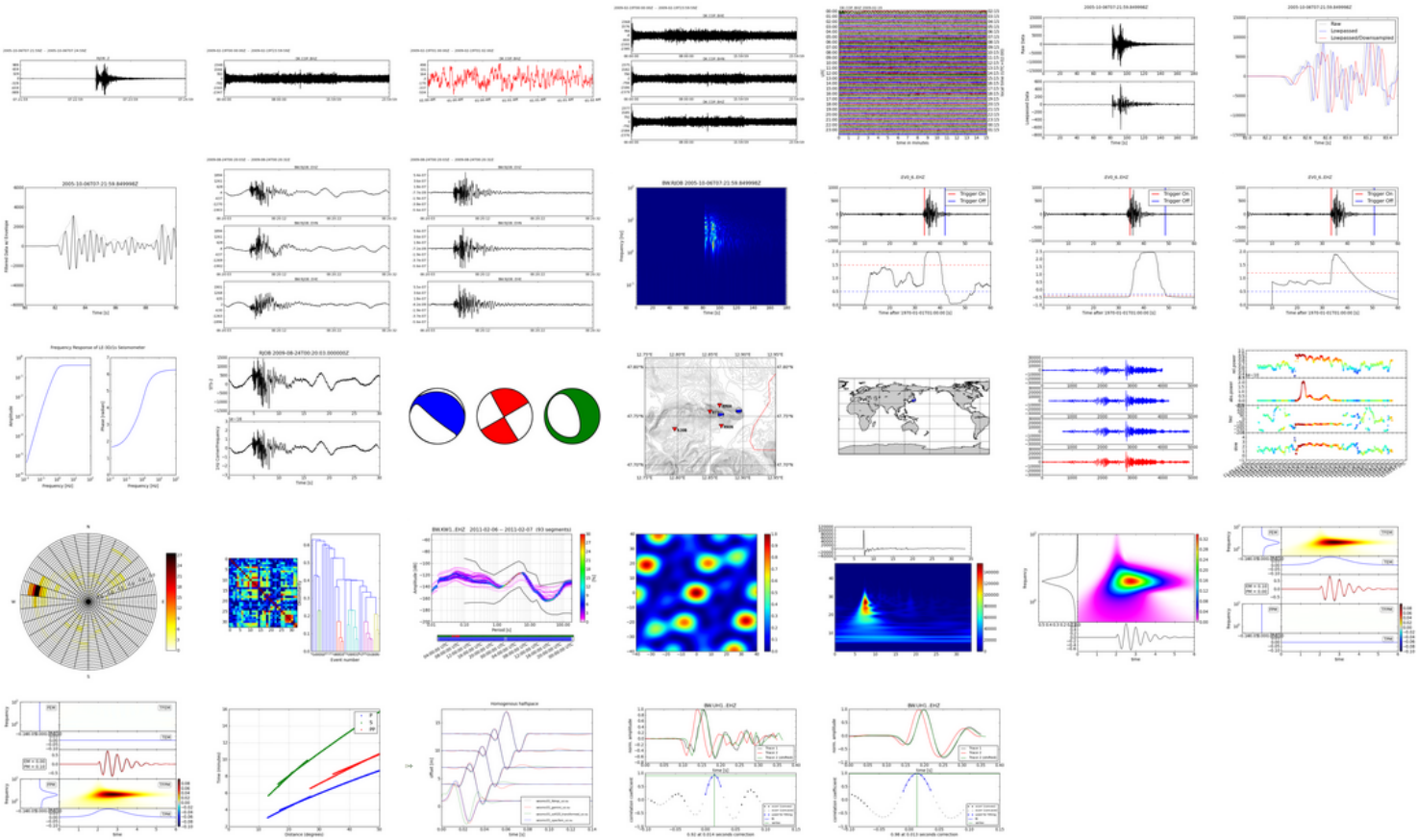
Quick search

Go

Enter search terms or a module, class or function name.

Gallery

Click on any image to see full size image and source code



Installing ObsPy

- **Debian/Ubuntu** - via package management (http://deb.obspy.org)
- **Windows** - Windows installer (automatically installs all dependencies)
- **OSX $\geq$ 10.6** - one-click-install application (contains all dependencies)

For the most recent additions and bug fixes install the developer version.

Detailed instructions for all platforms can be found on **www.obspy.org**.

Thanks for your attention.