# ObsPy's Core Functionality

## ObsPy Workshop

ObsPy Developers

SED/ETHZ

Zurich, Sept 6-8 2012

# obspy.core

This central module is the glue between all other ObsPy modules.

- Unified interface and functionality for handling waveform data in form of the **Stream** and **Trace** classes.
- All absolute time values within ObsPy are consistently handled with the **UTCDateTime** class.
- Event data is handled with the **Event** class. More about this later on.
- Generally useful utility classes and functions like the **AttribDict** class.
- Management via plugin discovery and binding, a global test script, . . .

# Goal: Familiarize Yourself With ObsPy's Core Objects and Functions

# 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

# Features of UTCDateTime

- Initialization

```
>>> from obspy.core import UTCDateTime
>>> UTCDateTime("2012-09-07T12:15:00")
UTCDateTime(2012, 9, 7, 12, 15)
>>> UTCDateTime(2012, 9, 7, 12, 15, 0)
UTCDateTime(2012, 9, 7, 12, 15)
>>> UTCDateTime(1347020100.0)
UTCDateTime(2012, 9, 7, 12, 15)
```

- Time zone support

```
>>> UTCDateTime("2012-09-07T12:15:00+02:00")
UTCDateTime(2012, 9, 7, 10, 15)
```

# Features of UTCDateTime

- Attribute access

```
>>> time = UTCDateTime("2012-09-07T12:15:00")
>>> time.year
2012
>>> time.julday
251
>>> time.timestamp
1347020100.0
>>> time.weekday
4
```

# Features of UTCDateTime

- Handling time differences

```
>>> time = UTCDateTime("2012-09-07T12:15:00")
>>> print time + 3600
2012-09-07T13:15:00.000000Z
>>> time2 = UTCDateTime(2012, 1, 1)
>>> print time - time2
21644100.0
```

# UTCDateTime - Exercises

1. Calculate the number of hours passed since your birth.
    - The current date and time can be obtained with **“UTCDateTime()”**
    - Optional: Include the correct time zone
2. Get a list of 10 UTCDateTime objects, starting yesterday at 10:00 with a spacing of 90 minutes.
3. The first session starts at 09:00 and lasts for 3 hours and 15 minutes. Assuming we want to have the coffee break 1234 seconds and 5 microseconds before it ends. At what time is the coffee break?
4. Assume you had your last cup of coffee yesterday at breakfast. How many minutes do you have to survive with that cup of coffee?

# Handling Waveform Data

```
>>> from obspy.core 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, \dots , -489, -339], dtype=int32)
```

# The Stream Object

- A **Stream** object is a collection of **Trace** objects

```
>>> from obspy.core import read
>>> st = read()
>>> type(st)
obspy.core.stream.Stream
>>> print st
3 Trace(s) in Stream:
BW.RJOB..EHZ | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
BW.RJOB..EHN | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
BW.RJOB..EHE | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
>>> st.traces
[<obspy.core.trace.Trace at 0x1017c8390>, ...]
>>> print st[0]
BW.RJOB..EHZ | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
>>> type(st[0])
obspy.core.trace.Trace
```

# The Trace Object

- A **Trace** object is a single, continuous waveform data block
- It furthermore contains a limited amount of metadata

```
>>> tr = st[0]
>>> print tr
BW.RJOB..EHZ | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
>>> print tr.stats
         network: BW
         station: RJOB
        location:
         channel: EHZ
       starttime: 2009-08-24T00:20:03.000000Z
         endtime: 2009-08-24T00:20:32.990000Z
   sampling_rate: 100.0
           delta: 0.01
            npts: 3000
           calib: 1.0
```

# The Trace Object

- For custom applications it is often necessary to directly manipulate the metadata of a Trace.
- The state of the Trace will stay consistent, as all values are derived from the starttime, the data and the sampling rate and are updated automatically.

```
>>> print tr.stats.delta, tr.stats.endtime
0.02 2009-08-24T00:20:27.980000Z
>>> tr.stats.sampling_rate = 5.0
>>> print tr.stats.delta, tr.stats.endtime
0.2 2009-08-24T00:23:27.800000Z
>>> print tr.stats.npts
3000
>>> tr.data = tr.data[:100]
>>> print tr.stats.npts, tr.stats.endtime
100 2009-08-24T00:20:27.800000Z
```

# The Trace Object

- Working with them is easy, with a lot of attached methods.

```
>>> print tr
BW.RJOB..EHZ | 2009-08-24T00: ... | 100.0 Hz, 3000 samples
>>> tr.resample(sampling_rate=50.0)
>>> print tr
BW.RJOB..EHZ | 2009-08-24T00: ... | 50.0 Hz, 1500 samples
>>> tr.trim(tr.stats.starttime + 5, tr.stats.endtime - 5)
>>> print tr
BW.RJOB..EHZ | 2009-08-24T00: ... | 50.0 Hz, 500 samples
>>> tr.detrend("linear")
>>> tr.filter("highpass", freq=2.0)
```

# Stream Methods

- Most methods that work on a **Trace** object also work on a **Stream** object. They are simply executed for every trace.
  - **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")
```

# Waveform Data - Exercises

Later on a useful example application will be developed. For now the goal is to get to know the Stream and Trace classes.

Several possibilies to obtain a Stream object:

- The empty **read()** method will return some example data.
- Passing a filename to the **read()** method.
- Using one of the webservices. This will be dealt with in the next part.
- Passing a URL to **read()**. See e.g. *examples.obspy.org/* for some files.

# Trace Exercise 1

- Make a trace with all zeros (e.g. *numpy.zeros(200)*) and an ideal pulse at the center
- Fill in some station information (*network*, *station*)
- Print trace summary and display the preview plot of the trace
- Change the sampling rate to 20Hz
- Change the *starttime* to the start time of this sessions
- Print trace summary and display the preview plot of the trace again

# Trace Exercise 2

- Use *tr.filter(...)* and apply a lowpass with 1s corner frequency
- Display the preview plot, there are a few seconds of zeros that we can cut off
- Use *tr.trim(...)* to remove some of the zeros at start and end

# Trace Exercise 3

- Scale up the amplitudes of the trace by a factor of 500
- Make a copy of the original trace
- Add standard normal gaussian noise to the copied trace (use *numpy.random.randn(..)*)
- Change the station name of the copied trace
- Display the preview plot of the new trace

# Stream Exercise 1

- read the example earthquake data into a stream object (*read()* without arguments)
- print the stream summary and display the preview plot
- assign the first trace to a new variable and then remove that trace from the original stream
- print the summary for the single trace and for the stream

# Stream Exercise 2

- Read the example earthquake data again
- Make a dictionary with paz information, assign poles at *[-0.037+0.037j, -0.037-0.037j]*, zeros at *[0j, 0j]*, the sensitivity of *2.517e9* and unity gain
- Remove the instrument response using this paz dictionary
- Print the data maximum and minimum of the first trace (now in m/s)
- Save the data to a local file in MSEED format

# Waveform Data - Exercises

Some further ideas what you can do now to get a better grasp of the objects:

1. Read some files from different sources and see what happens
2. Have a look at the ObsPy Documentation on the homepage
3. Use IPython's tab completion and help feature to explore objects