ObsPy: Station and Event Data Handling

ObsPy Workshop

ObsPy Developers

SED/ETHZ

Zurich, Sept 6-8 2012

Inventory Data - obspy.xseed

- 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>
...
```

obspy.xseed - 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]}
```

obspy.xseed - Example usage

```
>>> p.writeXSEED("dataless.xml")
# Edit it ...
>>> p = Parser("dataless.xml")
>>> p.writeSEED("edit_dataless_SEED")
```

obspy.xseed - Exercise

- Read the **BW.FURT..EHZ.D.2010.005** waveform example file.
- Cut out some minutes of interest.
- Read the **dataless.seed.BW_FURT** SEED file.
- Correct the trimmed waveform file with the poles and zeros from the dataless SEED file using *st.simulate()*. This will, according to the SEED convention, correct to m/s.
- (Optional) Read the file again and convert to m by adding an extra zero. Choose a sensible waterlevel.
- (Optional) Convert the SEED file to XSEED, edit some values and convert it back to SEED again. This requires some knowledge of the general SEED file structure.

Events - Work in progress

- 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.
- Currently only supports QuakeML and is modelled after it.

```
>>> from obspy.core.events 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  | ...
...
```

Events - Basic Structure

- The **readEvents()** function always returns a **Catalog** object, which is a collection of **Event** objects.

```
>>> from obspy.core.events import readEvents
>>> cat = readEvents()
>>> type(cat)
obspy.core.event.Catalog
>>> type(cat[0])
obspy.core.event.Event
```

Events - Basic Structure

```
>>> event = cat[0]
>>> print event
Event:  2012-04-04T14:...| +41.818,  +79.689 | 4.4 mb

          resource_id: ResourceIdentifier(...)
           event_type: "not reported"
        creation_info: CreationInfo
           agency_uri: ResourceIdentifier(...)
           author_uri: ResourceIdentifier(...)
        creation_time: UTCDateTime(2012, 4, 4, 16, 40, 50)
              version: "1.0.1"
      ---------
                  origins: 1 Elements
                magnitudes: 1 Elements
```

Events - Basic Structure

- **Event** objects are again collections of other resources.

```
>>> type(event.origins[0])
obspy.core.event.Origin
>>> type(event.magnitudes[0])
obspy.core.event.Magnitude
>>> print event.origins[0]
Origin
         resource_id: ResourceIdentifier(...)
                time: UTCDateTime(...)
            latitude: 41.818
           longitude: 79.689
               depth: 1.0
          depth_type: "from location"
           method_id: ResourceIdentifier(...)
  used_station_count: 16
       azimuthal_gap: 231.0
       ...
```

Events - Resource References

- In QuakeML resources can refer to each other using a unique identifier string.
- These connections are preserved in obspy.core.event.
- This works across file boundaries assuming all necessary resources have been read before.

```
>>> magnitude = event.magnitudes[0]
# Retrieve the associated Origin object.
>>> print magnitude.origin_id
quakeml:eu.emsc/origin/rts/261020/782484
>>> origin = magnitude.origin_id.getReferredObject()
>>> print origin
Origin
  resource_id: ResourceIdentifier(...)
         time: UTCDateTime(2012, 4, 4, 14, 21, 42, 300000)
     latitude: 41.818
    longitude: 79.689
    ...
```

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")
```

obspy.core.event - Exercise

- Get the 5 latest events, directly from the seimicportal.eu webservice.
  - **cat = readEvents(**
    "http://www.seismicportal.eu/services/event/latest?num=5")
- Plot the events.
- Print the resulting Catalog object and filter it, so it only contains the three largest events.
- Now assume you did a new magnitude estimation and want to add it to one event. Create a new magnitude object, fill it with some values and append it to magnitude list of the largest event.
- Write the Catalog as a QuakeML object.