1 APPENDIX: EXAMPLE OF TIME DATA MANIPULATION

This section shows how time data are handled in practice when working with Razorback, particularly how to inspect and organize the data and how to compute impedance.

We consider a fictional situation where one has recorded MT signals (Ex, Ey, Hx, Hy, Hz) on 5 sites (1, 2, 3, 4, 5), some with several runs and different sampling rates. The situation could be pictured as follows:

```
==== ======================================
site   time
==== ======================================
  1            ~~~~~~~~
  2    ~~~~~~   ~~~~~~      ^^^^^^^
  3     ~~~~~~~~~~~~~~~~  ^^^^^^^^^^
  4    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  5    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
==== ======================================


~~~~~ : continuous run sampled at 512 Hz
^^^^^ : continuous run sampled at 1024 Hz
```

Each channel of each run is stored in one file, meaning that 40 files are involved in this example. Razorback provides tools to load raw data from some file type and tools to infer tags from the file path. Here, we assume that the 40 files have been loaded and tagged in 40 `SignalSet` objects, all gathered in one list, `all_signals`.

First, we can inspect some elements of `all_signals`:

```
>>> print(len(all_signals))
40
>>> print(all_signals[0])
SignalSet: 1 channel, 1 run
tags: {'Ex_1': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:01:50  1970-01-01 00:04:10
----------  -------------------  -------------------
>>> print(all_signals[14])
SignalSet: 1 channel, 1 run
tags: {'Hz_2': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
```

```
----------  -------------------  -------------------
>>> print(all_signals[19])
SignalSet: 1 channel, 1 run
tags: {'Hz_2': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

Printing a `SignalSet` provides a short and readable report. Here, we see the only run of the Ex channel on site 1 and two runs of the Hz channel on site 2, one at 512 Hz and the other at 1024 Hz.

The `SignalSet` objects can be grouped along the same channel using the `|` operator or along the same run using the & operator:

```
>>> print(all_signals[14] | all_signals[19])
SignalSet: 1 channel, 2 runs
tags: {'Hz_2': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
>>> print(all_signals[0] & all_signals[14])
SignalSet: 2 channels, 1 run
tags: {'Ex_1': (0,), 'Hz_2': (1,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
----------  -------------------  -------------------
```

In the first case, the result has 1 channel ('Hz_2') and 2 runs. In the second case, the result has 2 channels ('Ex_1' and 'Hz_2') and 1 run; we can see that the start and stop of the run are adapted to include the largest common run.

Using the `|` and & operators, we could gather our data in different ways in preparation for different processings. This would be tedious, even for this small example. Using an `Inventory` object simplifies the manipulation. First, we create an inventory from the list of data:

```
>>> inv = Inventory(all_signals)
```

The inventory behaves similar to a list:

```
>>> print(len(inv))
40
>>> print(inv[19])
SignalSet: 1 channel, 1 run
tags: {'Hz_2': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

However, the inventory can also inspect its content; for instance, we can see all the tags defined in the inventory:

```
>>> print(inv.tags)
set(['Ey_2', 'Hy_2', 'Ex_1', 'Hz_1', 'Ey_3', 'Hy_1', 'Ey_1', 'Hy_3', 'Hy_4',
'Hy_5', 'Ey_5', 'Ey_4', 'Hx_1', 'Hx_3', 'Hx_2', 'Hx_5', 'Hx_4', 'Ex_2',
'Ex_3', 'Hz_5', 'Hz_4', 'Hz_3', 'Hz_2', 'Ex_4', 'Ex_5'])
```

Using the `pack()` method, the inventory builds a `SignalSet` gathering all its content. However, this operation is not always possible due to the strict structure of a `SignalSet`. If we perform `pack()` on the entire inventory, we receive nothing (`None` in Python):

```
>>> print(inv.pack())
None
```

Instead, we first have to extract a consistent part of the inventory. The `filter()` method is designed for this task. To obtain a new inventory containing the `'Ex_2'` channel data, we perform the following:

```
>>> inv_Ex_2 = inv.filter('Ex_2')
>>> len(inv_Ex_2)
3
>>> print(inv_Ex_2[0])
SignalSet: 1 channel, 1 run
tags: {'Ex_2': (0,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:00:00  1970-01-01 00:01:40
----------  -------------------  -------------------
>>> print(inv_Ex_2[1])
SignalSet: 1 channel, 1 run
tags: {'Ex_2': (0,)}
----------  -------------------  -------------------
```

```
  sampling                 start                  stop
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
----------  -------------------  -------------------
>>> print(inv_Ex_2[2])
SignalSet: 1 channel, 1 run
tags: {'Ex_2': (0,)}
----------  -------------------  -------------------
  sampling                 start                  stop
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

This new inventory can be packed into one `SignalSet`:

```
>>> print(inv_Ex_2.pack())
SignalSet: 1 channel, 3 runs
tags: {'Ex_2': (0,)}
----------  -------------------  -------------------
  sampling                 start                  stop
       512  1970-01-01 00:00:00  1970-01-01 00:01:40
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

The `filter()` method accepts flexible patterns on tags, allowing, for instance, to build the `SignalSet` of one site:

```
>>> print(inv.filter('*_2').pack())
SignalSet: 5 channels, 3 runs
tags: {'Ex_2': (0,), 'Ey_2': (1,), 'Hx_2': (2,),
       'Hy_2': (3,), 'Hz_2': (4,)}
----------  -------------------  -------------------
  sampling                 start                  stop
       512  1970-01-01 00:00:00  1970-01-01 00:01:40
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

Multiple patterns can be passed; thus, gathering sites 2 and 3 is performed as follows:

```
>>> print(inv.filter('*_2', '*_3').pack())
SignalSet: 10 channels, 3 runs
tags: {'Ex_2': (0,), 'Ex_3': (1,), 'Ey_2': (2,),
       'Ey_3': (3,), 'Hx_2': (4,), 'Hx_3': (5,),
```

```
       'Hy_2': (6,), 'Hy_3': (7,), 'Hz_2': (8,),
       'Hz_3': (9,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:00:10  1970-01-01 00:01:40
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

In addition to the `filter()` method, the inventory provides the `select_runs()` method. This method can be used to group the data according to the sampling rates:

```
>>> inv_512 = inv.select_runs(f == 512 for f in inv.sampling_rates)
>>> inv_1024 = inv.select_runs(f == 1024 for f in inv.sampling_rates)
```

Packing these inventories produces the largest common part of the data for each sampling rate:

```
>>> print(inv_512.pack())
SignalSet: 20 channels, 1 run
tags: {'Ex_1': (0,), 'Ex_2': (1,), 'Ex_3': (2,),
       'Ex_4': (3,), 'Ey_1': (4,), 'Ey_2': (5,),
       'Ey_3': (6,), 'Ey_4': (7,), 'Hx_1': (8,),
       'Hx_2': (9,), 'Hx_3': (10,), 'Hx_4': (11,),
       'Hy_1': (12,), 'Hy_2': (13,), 'Hy_3': (14,),
       'Hy_4': (15,), 'Hz_1': (16,), 'Hz_2': (17,),
       'Hz_3': (18,), 'Hz_4': (19,)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
----------  -------------------  -------------------
>>> print(inv_1024.pack())
SignalSet: 15 channels, 1 run
tags: {'Ex_2': (0,), 'Ex_3': (1,), 'Ex_5': (2,),
       'Ey_2': (3,), 'Ey_3': (4,), 'Ey_5': (5,),
       'Hx_2': (6,), 'Hx_3': (7,), 'Hx_5': (8,),
       'Hy_2': (9,), 'Hy_3': (10,), 'Hy_5': (11,),
       'Hz_2': (12,), 'Hz_3': (13,), 'Hz_5': (14,)}
----------  -------------------  -------------------
  sampling                start                 stop
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

If we want to calculate the impedance on site 2 using sites 3 and 4 as RRs, we can use the `impedance` function (see section 3.4). This requires gathering in one `SignalSet` the electric and magnetic channels from site 2 and the magnetic channels from sites 3 and 4, as well as adding tags indicating which channels must be used as output, input or remote. Since two sampling rates (512 Hz and 1024 Hz) are involved on these sites with only partial overlapping of the runs, we must treat each sampling rate separately. The following shows how to do so for the 512 Hz sampling rate:

```
>>> sig_2_512 = inv_512.filter('[EH][xy]_2', 'H[xy]_3', 'H[xy]_4').pack()
>>> t = sig_2_512.tags
>>> t['E'] = t['Ex_2'] + t['Ey_2']
>>> t['H'] = t['Hx_2'] + t['Hy_2']
>>> t['Hremote'] = t.filter_get('H[xy]_3', 'H[xy]_4')
>>> print(sig_2_512)
SignalSet: 8 channels, 2 runs
tags: {'Ex_2': (0,), 'Ey_2': (1,), 'Hx_2': (2,),
       'Hx_3': (3,), 'Hx_4': (4,), 'Hy_2': (5,),
       'Hy_3': (6,), 'Hy_4': (7,), 'H': (2, 5),
       'E': (0, 1), 'Hremote': (3, 4, 6, 7)}
----------  -------------------  -------------------
  sampling                start                 stop
       512  1970-01-01 00:00:10  1970-01-01 00:01:40
       512  1970-01-01 00:02:00  1970-01-01 00:03:40
----------  -------------------  -------------------
```

Here, we used the `tags` attribute of the `SignalSet` (`t = sig_2_512.tags`) to add the new tag names `'E'`, `'H'` and `'Hremote'`. We can duplicate the same code for the 1024 Hz sampling rate:

```
>>> sig_2_1024 = inv_1024.filter('[EH][xy]_2', 'H[xy]_3', 'H[xy]_4').pack()
>>> t = sig_2_1024.tags
>>> t['E'] = t['Ex_2'] + t['Ey_2']
>>> t['H'] = t['Hx_2'] + t['Hy_2']
>>> t['Hremote'] = t.filter_get('H[xy]_3', 'H[xy]_4')
>>> print(sig_2_1024)
SignalSet: 6 channels, 1 run
tags: {'Ex_2': (0,), 'Ey_2': (1,), 'Hx_2': (2,),
       'Hx_3': (3,), 'Hy_2': (4,), 'Hy_3': (5,),
       'B': (2, 4), 'Hremote': (3, 5), 'E': (0, 1)}
----------  -------------------  -------------------
  sampling                start                 stop
      1024  1970-01-01 00:06:40  1970-01-01 00:07:30
----------  -------------------  -------------------
```

Note that the lack of data sampled at 1024 Hz on site 4 does not prevent building the `SignalSet` and that the `'Hremote'` tag is still correctly initialized. Once they are correctly gathered and tagged, computing the impedance at some frequencies (1 Hz, 8 Hz and 16 Hz in the example) using the two-stage RR method with a least-squares estimator is performed as follows:

```
>>> result = impedance(sig_2_512, [1, 8, 16], remote='Hremote')
```