Powering up your FME Workspaces with Python

Exercise Handout

Instructors:

Tino Miegel t.miegel@conterra.de

Dennis Wilhelm d. wilhelm@conterra.de

Help

If you are stuck at any exercise, ask your instructor for help or have a look at the solution workspaces at

C:\FMEData\Workspaces\Python

Exercise 1 – Python Basics

Learning Goal: Test a few basic Python commands using the FME Python interpreter.

Steps:

1. Open a command line window and start a Python shell using: `fme.exe python`
2. Enter the following commands and inspect the results in the console window
   a. `1+1`
   b. `“1” * 5`
   c. `dir()`
   d. `values = [1,2,3,4]`
   e. `print(values)`
3. Enter some commands into the file (one per line)

Exercise 2 – Python Startup Script

Learning Goal: Use Python Startup Scripts and the fmeobjects logging function.

Steps:

1. Create a new Workspace and open the Python Startup Script (Navigator -> Workspace Parameters -> Scripting -> Startup Python Script)
2. Import the `fmeobjects` module.
   a. The API documentation can be found on https://docs.safe.com/fme/html/FME_Objects_Python_API/index.html
3. Create a new instance of the FMELogFile class.

```
myLogger = fmeobjects.FMELogFile()
```

4. Use the API documentation to find out how to use the function `logMessageString` from the `FMELogFile` class.
5. Write three log messages with different severity levels in your startup script.

   Run the workspace and inspect the log file. The log messages should be right after the message: *FME_BEGIN_PYTHON: evaluating python script from string...*

Exercise 3 – Calculate statistics

Learning Goal: Use all the different combinations PythonCaller methods to aggregate multiple FME features.

Input Data: CellSignal.csv

Final Workspace: statistics.fmw

Steps:

1. Open a new Workspace.

2. Add a new CSV Reader to read the following file:
   - `C:\FMEData\Data\CellSignals\CellSignal.csv`

3. View the file using the data inspector.

4. Add a PythonCaller after the Reader. We are going to use the Class implementation which is already provided as a template called FeatureProcessor.

5. Tip: During development, use a Sampler Transformer in front of the PythonCaller to reduce the number of features.

   First, define a variable `self.qualities` within the `init()` method that will be used to cache attribute values for each feature.
   ```
   self.qualities = []
   ```

6. The `input()` method is called for each feature (each row in the CSV file). Use this method to read the "Quality" attribute of each feature and store it in the previously defined list.
   ```
   self.qualities.append(feature.getAttribute('Quality'))
   ```

7. The `close()` method is automatically executed when all previous features have been processed. Now the statistics can be calculated here. Use the `min/max/sum` functions to calculate simple statistics.
   ```
   total = sum(self.qualities)
   ```

8. To create a new FME Feature use the following call:
   ```
   statisticFeature = fmeobjects.FMEFeature()
   ```

9. Now store the new values as attributes on the feature.
   ```
   statisticFeature.setAttribute('Sum', total)
   ```

10. Finally, return the generated feature to the FME process
    ```
    self.pyoutput(statisticFeature)
    ```

11. Check the result in the Data Inspector.

12. Optional: consider how to group the data by the "StationID" attribute.

Exercise 4 – Working with list attributes

Learning Goal: Editing FME list attributes with Python. Within the source file there are list entries with measured values and the unit (cm). Due to the unit, the value cannot be used in calculations (e.g. ListSummer transformer). In the task, the list entries are to be cleaned up by Python.

Input Data: waterlevel_list.ffs

Final Workspace: ListHandling.fmw

Steps:

1. Open a new workspace

2. Add a new Reader and have a look at the data in the visual preview window:
   Format: **FME Feature Store (FFS))**
   File: **waterlevel_list.ffs**

3. Add a PythonCaller behind the Reader.

4. Create a for-loop within the input method to iterate over the individual measurements of the attribute list `measurements{}.measurement`. You can decide if you want to use the `enumerate`-method or a custom counter variable.

5. Replace the unit (cm) within the list element using the `replace`-method and store the result in a new variable.

6. Overwrite the current list element within the loop using the new value:
   `feature.setAttribute(measurements{'+str(i)+'}.measurement, newValue)`

7. The new list can now be used in the default list transformers. Add the following three transformers behind the PythonCaller: `ListElementCounter, ListSummer, ExpressionEvaluator`.

8. Calculate the mean value of all measurements using the `ExpressionEvaluator` (Sum/Count of list elements).

9. Have a look at the result.

Exercise 5 – Use the FME Connection Manager

Learning Goal: Use the new FME Webservice API in Python to access user credential from a FME Web Connection. To validate the values, we are going to use the Python library requests and http://httpbin.org/ , which allows testing HTTP Basic Authentication with arbitrary self chosen credentials.

Final Workspace: Exercise_5_End.fmw

Input Data: None

Steps:

1. Open the FME Web Connections via Tools -> FME Options
2. Create a “HTTP Authentication” with these values:
    a. Connection Name: Test Connection
    b. User Name: fmeuser
    c. Password: fmepassword
    d. Authentication Method: Basic
3. Add a PythonCreator Transformer to the canvas.
    a. Modify the code template and add two imports at the top

```
import fmewebservices
import requests
```

4. In the input method add:

```
conn_man = fmewebservices.FMENamedConnectionManager()
conn = conn_man.getNamedConnection('Test Connection')
```

5. Now you can request username and password from the connection object

```
username = conn.getUserName()
password = conn.getPassword()
```

6. Use username and password to make a connection

```
response = requests.get('http://httpbin.org/basic-
auth/fmeuser/fmepassword',auth=(username, password))
```

6. Now create a feature with the response data

```
newFeature = fmeobjects.FMEFeature()
newFeature.setAttribute('status_code', response.status_code)
newFeature.setAttribute('body_text', response.text)
self.pyoutput(newFeature)
```

Run the workspace and check if the status_code is 200. The HTTPbin test endpoint work simply with /<username>/<password> so it validates the credentials sent via HTTP Basic against the URL. If you change either the URL or the credentials stored in your FME Web Connection, you should be able to produce a "401 – Unauthorized".

Of course, there is no need apart from demonstration purposes to use Python requests for simple HTTP calls, but a real life use case could be "client certificate authentication", a authentication method the HTTPCaller does allow at this point.