THE PEAK OF DATA INTEGRATION 2022 UC

con•terra

Powering up your FME Workspaces with Python

SAFE SOFTWARE®

Your Trainers

Tino Miegel

Dennis Wilhelm

con terra

CERTIFIED Professional

CERTIFIED Server

CERTIFIED Trainer

Agenda

- Introduction to Python
- Python and FME
  - Scripted Parameter
  - Startup/ Shutdown Scripts
  - Python Transformer
  - FME Connection Manager
  - Handling list attributes
  - Using group by
- Outlook

Environment & Materials

- Remote Desktop Image
  - FME Desktop 2022
  - Training-Data
- Exercise Handout
- FME Python Cheat Sheet

Training Image

Virtual Machine:
If you haven’t set up a VM, please go to
http://fme.ly/ucvm

Username: administrator
Password: FME2022learnings

Let's explore FME and Python

What is Python?

- Python is a scripting language.
  - Object oriented
  - No compiling or linking
  - Fast (“quick & dirty”) programming and prototyping
- Name: Developer van Rossum is a huge fan of Monty Python’s Flying Circus

Why Python?

- Free, powerful and flexible
- Platform independent
- Automatic Garbage Collecting
- Capable of being integrated
  - e.g. FME, ArcGIS, Blender
- Extensive documentation
  - www.python.org

Time for some actual Python

Python Basics

- Get a python shell with

```
#> <FME dir>/fme.exe python
```

- Run a script with

```
#> <FME dir>/fme.exe my_script.py
```

- Basics

```
> 1+1
> "1" * 5
> dir()
> values = [1,2,3,4,5]
> print(values)
```

Exercise 1

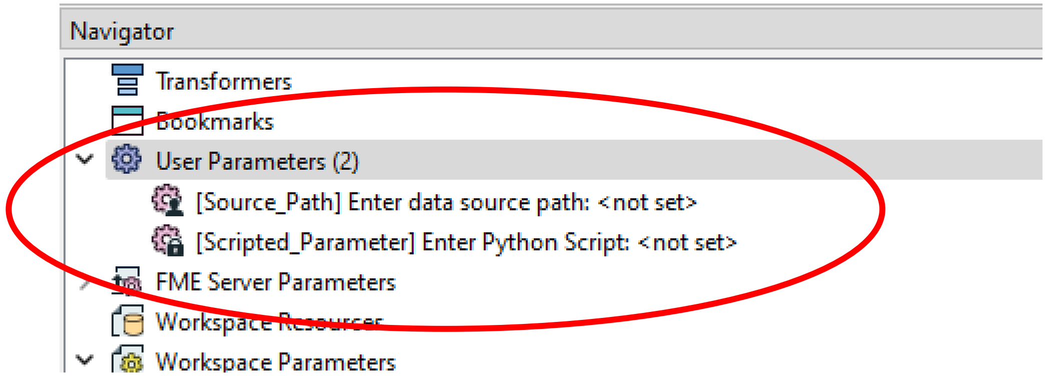
Setup and Basics

Connect to your personal trainings instance

Try out some basic Python commands

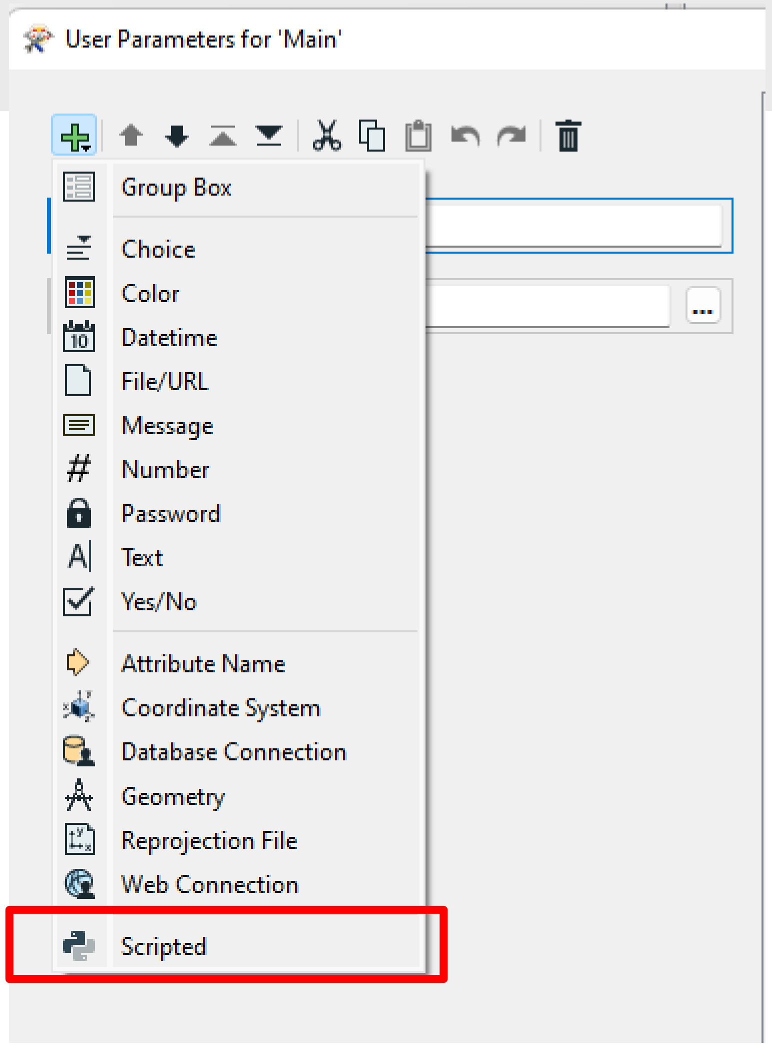
Published Parameters

- Published Parameter are parameters which are set before runtime.
- Examples:
  - Reader- / Writer file sources/ targets
  - Coordinate systems
  - Transformer parameters
  - Workspace Settings
  - Logfile


Navigator
Transformers
Bookmarks
User Parameters (2)
[Source_Path] Enter data source path: <not set>
[Scripted_Parameter] Enter Python Script: <not set>
FME Server Parameters
Workspace Resources
Workspace Parameters

Scripted Parameter

- Value is either Python or TCL script
- Order of execution:
  - Scripted Parameter
  - Startup Script
  - FME Process
  - Shutdown Script
- Allows usage as Reader parameter



Scripted Parameter

- Last line with `return` statement to hand value to FME process
- Access of Published Parameter
  - New: `fme.macroValues[‘Parameter_Name‘]`
  - Old: `FME_MacroValues[‘Parameter_Name‘]`

Demo

Scripted Parameter / INI File

Demo:

Use Python Parameters to read configs from an INI file.

FME Objects & Plugin API

- FME Objects API
  - Library containing FME functionality
- Plugin API
  - Develop Readers, Writers, Formats
  - Uses FME Objects
- Documentation
  - https://docs.safe.com/fme/html/fmepython/index.html
    - Python FME Objects / Python FME Webservices API
  - https://safe.com/documentation

fmeobjects

- Central Module `fmeobjects`
- Code statement: `import fmeobjects`
- Many classes for different FME aspects:
    - *`FMEFeature, FMEGeometry, FMELogfile, …`*

FMELogFile()

- Create your own log messages (also on FMEServer!)
- Create a logger object
- `logger = fmeobjects.FMELogfile()`

- Don't forget to import fmeobjects
- Create a log message
- `logger.logMessageString(message, severity)`

Using print()

- `print('Info message',[file=sys.stdout])`
- `print('Warn message',[file=sys.stderr])`
- => Only for rapid debugging
- Use `fmeobjects.FMELogFile()` optimal

Severity Types

Optional: Log-Level (FME Severity Level)

```
self.logger.logMessageString('Message', fmeobjects.FME_WARN)
```

0	FME_INFORM	black
1	FME_WARN	blue
2	FME_ERROR	red
…		

Exercise 2

Python Startup Script

Problem:

To clarify things you want to add your own custom log messages to the FME Logfile.

Solution:

Use the logger facility FMELogFile() and create messages with different log levels.

Shutdown Script

- Runs after the process has finished with either SUCCESS or FAILURE
- Post-Processing
  - Everything FME related is done by then
- Use cases
  - Move / copy / pack result
  - Call external modules (e.g. arcpy)
  - Custom logging

Shutdown Script

- Access published parameters and FME system parameters with the module fme
  - `fme.cpuTime, fme.cpuUserTime, fme.featuresRead, fme.failureMessage, fme.logFileName, fme.macroValues, fme.status,` …
  - All shutdown script variables
    - https://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Desktop/Configuration/FME_END_PYTHON.htm
- You can't use `fmeobjects.FMELogfile()` !
  - Simple workaround:

```
with open(fme.logFileName, "a") as logfile:
     logfile.write("Processing Shutdown Script\n")
```

fmeobjects - FME Feature

```
import fmeobjects

# Instantiate a new feature

myFeature = fmeobjects.FMEFeature()
```

fmeobjects - FME Feature

```
myFeature.setAttribute("Identification", 123)

myFeature.setAttribute("Name", "FME Lizard")

myFeature.setAttribute("List", ["FME Desktop", "FME Server"])
```

fmeobjects - FME Feature

```
myFeature.removeAttribute("Name")

myFeature.removeAttrsWithPrefix("Any Prefix")
```

Working with geometries

- Two steps to create a feature with a geometry
  - Create geometry
  - Apply geometry to a FME feature

Step 1:

- Create a point geometry

```
point = fmeobjects.FMEPoint(0,0)
```

- Create a line geometry

```
line = fmeobjects.FMELine()
line.appendPoints([(-20,-20),(20,-20)])
```

Working with geometries

Step 2: Assign geometry to Feature

```
feature = fmeobjects.FMEFeature()
feature.setGeometry(point)

feature2 = fmeobjects.FMEFeature()
feature2.setGeometry(point)
```

More Feature Functions

- `getAllCoordinates()`
- `getGeometryType()`
- `getDimension()`
- `getCoordSys()`
- `...`

Coffee Break

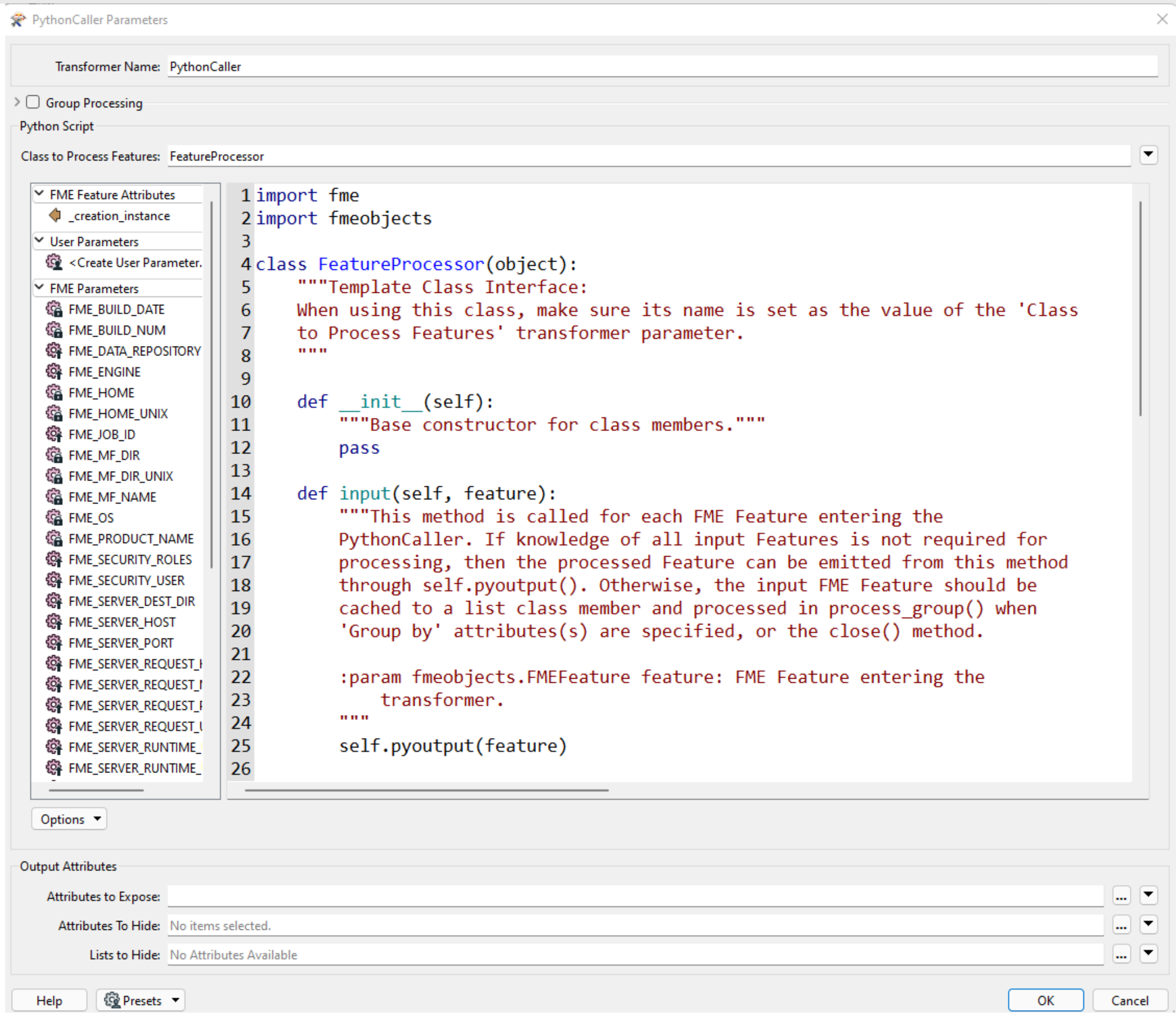
Python Transformer

- Both Transformers allow Python code execution during the FME process.
- Implement your code
  - Directly in Transformer
  - As external script file, e.g. myPythonLogic.py
- Use the PythonCaller to manipulate existing features (has an input port)
- Use PythonCreator the create features from scratch

Python Transformer

Variant A: Use FME Editor for source code

- FME Editor uses spaces indentation!
- Syntax-Highlighting
- Easy access of Parameters (Published, Private, System) Search & Replace
- But: No IntelliSense!



Transformer Name: PythonCaller

Class to Process Features: FeatureProcessor

```python
import fme
import fmeobjects

class FeatureProcessor(object):
    """Template Class Interface:
    When using this class, make sure its name is set as the value of the 'Class
    to Process Features' transformer parameter.
    """

    def __init__(self):
        """Base constructor for class members."""
        pass

    def input(self, feature):
        """This method is called for each FME Feature entering the
        PythonCaller. If knowledge of all input Features is not required for
        processing, then the processed Feature can be emitted from this method
        through self.pyoutput(). Otherwise, the input FME Feature should be
        cached to a list class member and processed in process_group() when
        'Group by' attributes(s) are specified, or the close() method.

        :param fmeobjects.FMEFeature feature: FME Feature entering the
            transformer.
        """

        self.pyoutput(feature)

```

Attributes to Expose:

Attributes To Hide: No items selected.

Lists to Hide: No Attributes Available

Python Transformer

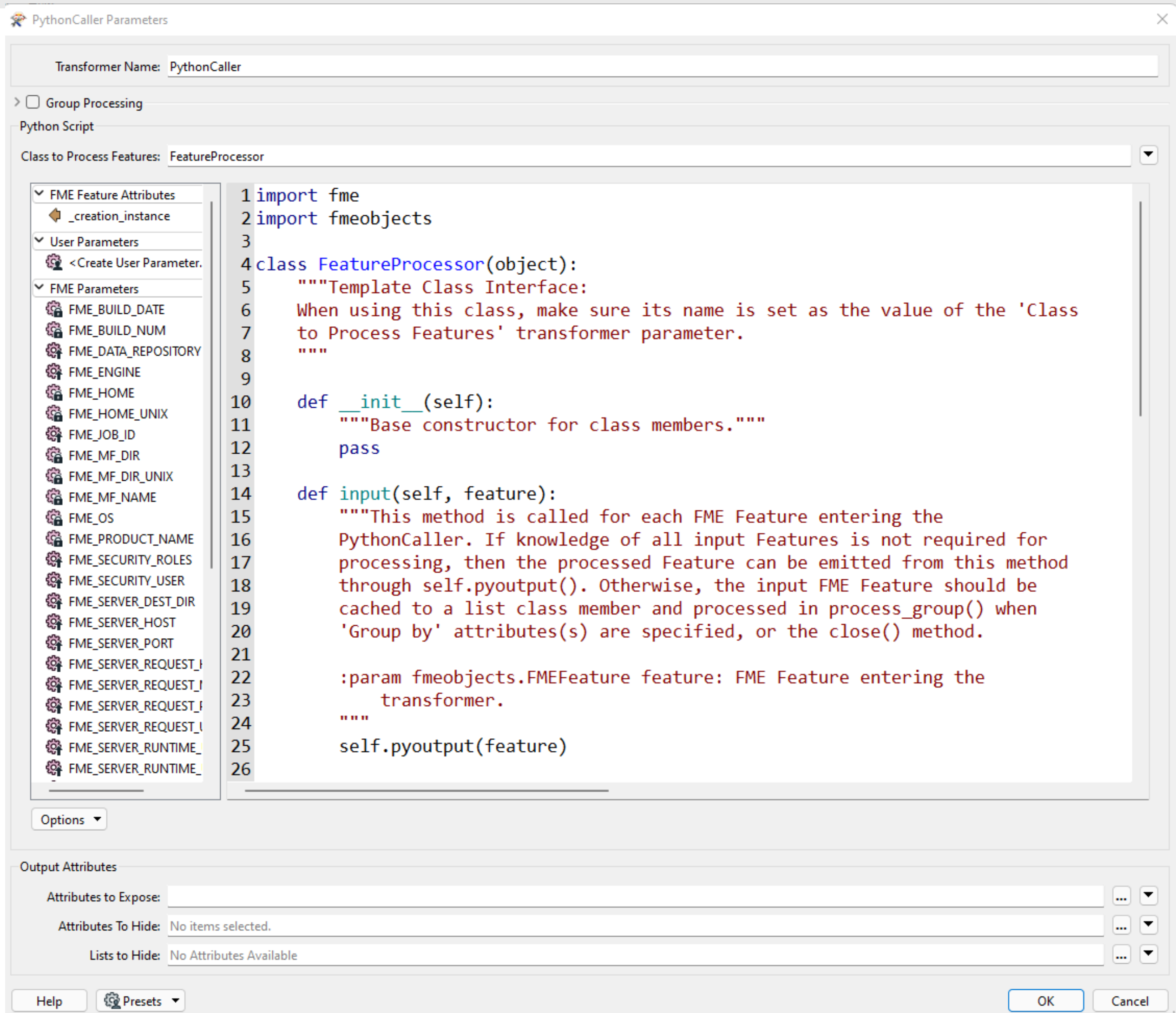
Variant B: External Script Files

- Class to process features `modulename.Class` => modulename.py
    - Benefit: You can use your favorite editor or IDE
    - Search path:

      ```
      <fme_Install>\FME\transformers
      <fme_Install>\FME\python
      ```
    - Directory of workspace file (*.fmw) ($FME_MF_DIR)
    - Add your own dirs with `sys.path.append` ! -> Startup Script

Python Transformer

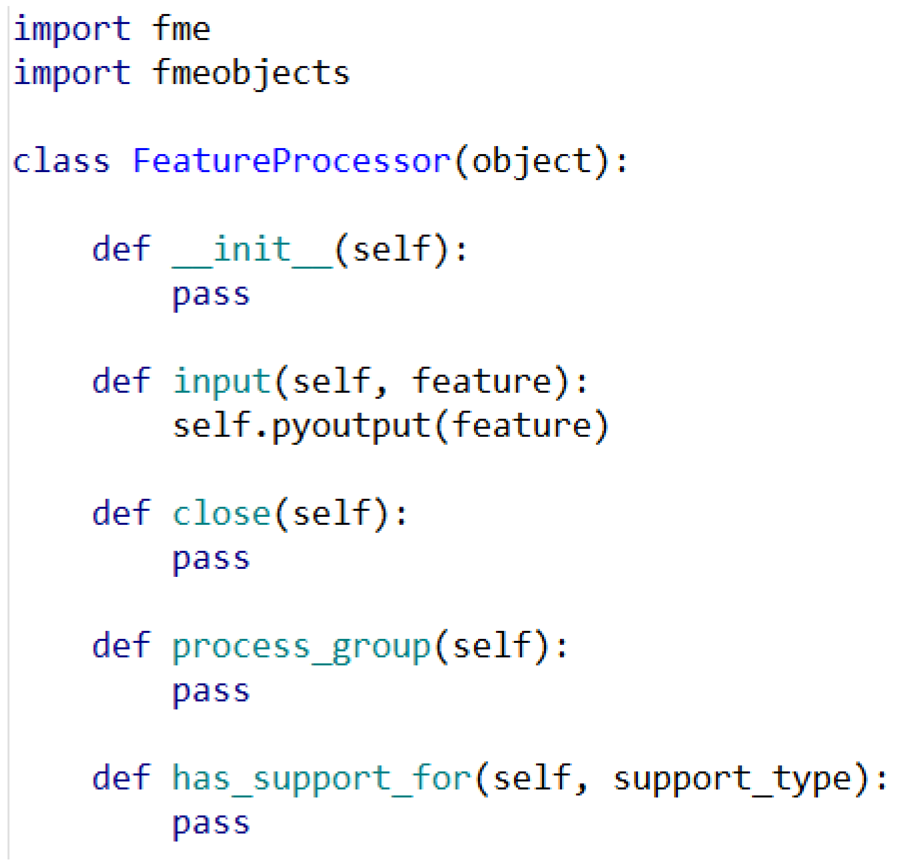
- Python Script
  - Pythonskript-/ Code
- Class to Process Features
- Attributes to Expose
- Attributes to Hide
- Lists to Hide



```python
import fme
import fmeobjects

class FeatureProcessor(object):
    """Template Class Interface:
    When using this class, make sure its name is set as the value of the 'Class
    to Process Features' transformer parameter.
    """

    def __init__(self):
        """Base constructor for class members."""
        pass

    def input(self, feature):
        """This method is called for each FME Feature entering the
        PythonCaller. If knowledge of all input Features is not required for
        processing, then the processed Feature can be emitted from this method
        through self.pyoutput(). Otherwise, the input FME Feature should be
        cached to a list class member and processed in process_group() when
        'Group by' attributes(s) are specified, or the close() method.

        :param fmeobjects.FMEFeature feature: FME Feature entering the
            transformer.
        """
        self.pyoutput(feature)

```

Class

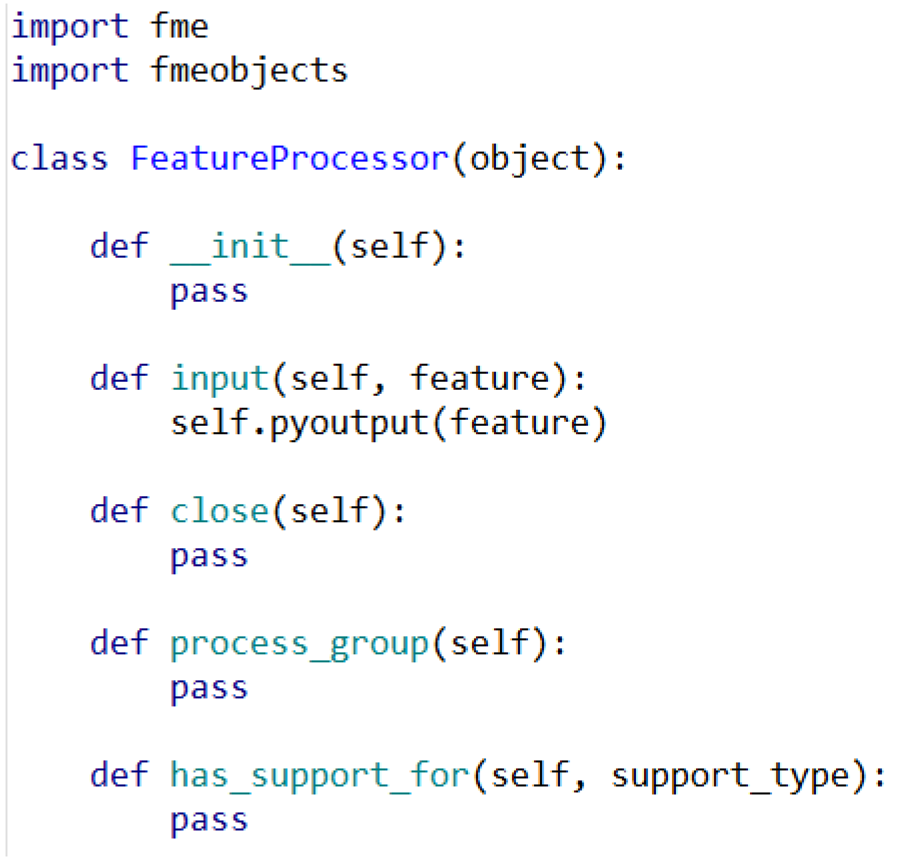
- Defined by the keyword `class`
- Constructor / function with reference to object
- FME hands over feature via `input(self,feature)`
- FME calls `close(self)` after last feature



```
import fme
import fmeobjects

class FeatureProcessor(object):

    def __init__(self):
        pass

    def input(self, feature):
        self.pyoutput(feature)

    def close(self):
        pass

    def process_group(self):
        pass

    def has_support_for(self, support_type):
        pass
```

Class

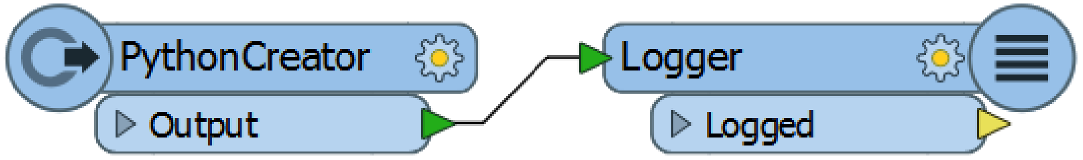
- Return feature handle to FME with `self.pyoutput()`
- Usable in both `input()` and `close()` method
  - Never access the feature object after `pyoutput()`!
  - Keyword `pass`
    - If a method is empty otherwise



```python
import fme
import fmeobjects

class FeatureProcessor(object):

    def __init__(self):
        pass

    def input(self, feature):
        self.pyoutput(feature)

    def close(self):
        pass

    def process_group(self):
        pass

    def has_support_for(self, support_type):
        pass
```

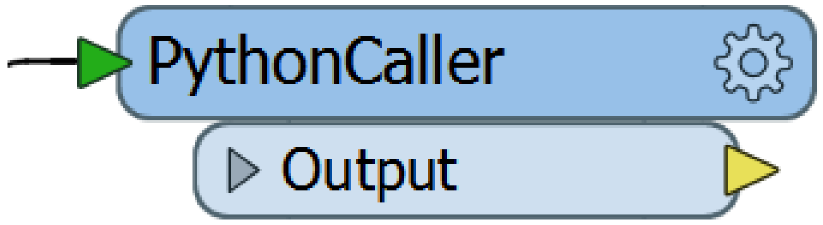
PythonCreator

- No input port!
- Usage
  - More control over creation of features compared to Creator transformer
  - Create your own Reader



PythonCaller

- Consumes FME features:
  - Attribute manipulation
  - Geometry manipulation
- Usage
  - Run any python code
  - Create advanced „Custom“ Transformers
  - Detailed logging, filtering or creation of features



Group by Processing

- Available in many transformer including PythonCaller
- Switches the transformer from „feature-by-feature“ to „feature-group-by-feature-group“ processing
- Can be memory intensive
- But avoids „FeatureFilter“ + multiple identical Transformers situations

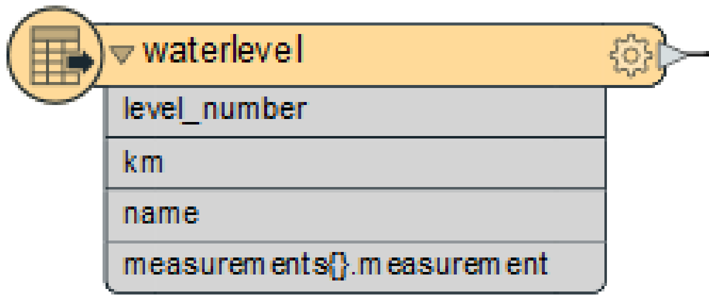
Exercise 3

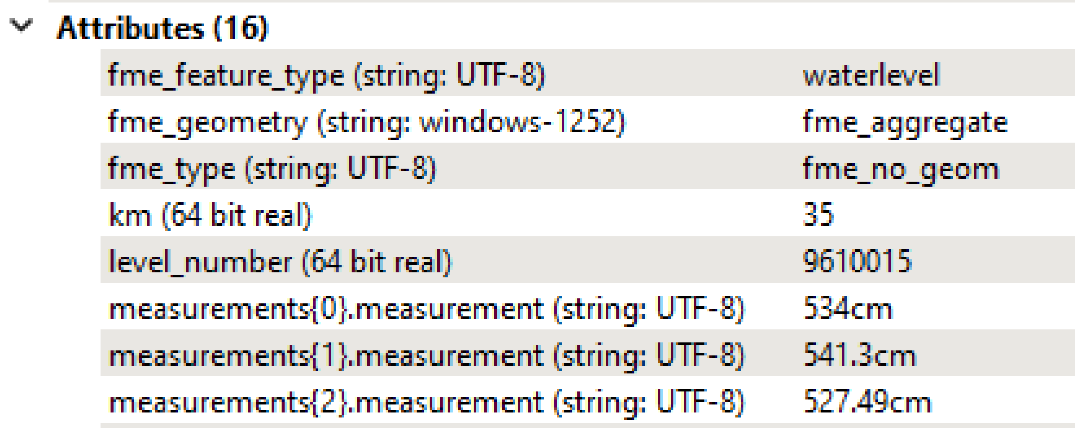
Calculate Statistics

Use different PythonCaller methods to calculate your own statistics over different features

Lists in FME

- Multiple values „in“ a single attribute per feature
- Note the difference
  - Notation via {} in FME
  - Notation via [] in Python
- setAttribute() / getAttribute() performs mapping





Attributes (16)

fme_feature_type (string: UTF-8)	waterlevel
fme_geometry (string: windows-1252)	fme_aggregate
fme_type (string: UTF-8)	fme_no_geom
km (64 bit real)	35
level_number (64 bit real)	9610015
measurements{0}.measurement (string: UTF-8)	534cm
measurements{1}.measurement (string: UTF-8)	541.3cm
measurements{2}.measurement (string: UTF-8)	527.49cm

FME Lists in Python

```
feature.getAttribute('measurements{}.measurement')
feature.setAttribute('measurements{}.measurement',[21,22,23])

feature.getAttribute('measurements{2}.measurement')
feature.setAttribute('measurements{2}.measurement', 123)

i = 2
feature.getAttribute('measurements{'+str(i)+'}.measurement')
```

Loops with FME Lists in Python

```
# „Normal“ for-Loop
myList = feature.getAttribute('_list{}._creation_instance')
for element in myList:
    print(element)


# Iteration with index
myList = feature.getAttribute('_list{}._creation_instance')
for i,element in enumerate(myList):
    print(i,element)
```

Exercise 4

Working with list attributes

Problem:

Non numeric list elements can’t be processed by some list transformers

Solution:

Use Python to iterate over the list elements and clean up the values

Named Connections

- Preferable storage of user credentials to external services
- Well integrated into FME Server
- Encrypted password storage
- Keeps confidential data out of workspace files

Exercise 5

Use the FME Connection Manager

Use the new FME Webservice API in Python to access user credential from a FME Web Connection

Using additional Libraries

- Check if already included
- User standalone Python interpreter and PIP
- Use FME PIP
- <Danger>
  - Version conflicts possible

Python Plugin SDK

- Samples and Documentation
  `<FMEHOME>\pluginbuilder`

Reader/Writer Plugin

- You'll need:
  - Your Code => `<FMEHOME>\plugins`
  - Formatsinfo File => `<FMEHOME>\formatsinfo`
  - Metafile
  - (Schema file)

Plugin Transformer

- FMX-File
- Pluginsinfo File
- Code

Python is a powerful way to extent FME

Don’t reinvent the wheel

Thank You!

Feel free to contact us during the conference!

t.miegel@conterra.de
d.wilhelm@conterra.de

THE PEAK OF DATA INTEGRATION 2022 UC

THANK YOU!

conterra.de

d.wilhelm@conterra.de | t.miegel@conterra.de

SAFE SOFTWARE®