

Milestone 1: Power System Equipment Classes

Joseph Rivera and Michael Bliesath

Keywords: Bus, Generator, Load, Transformer, Transmission Line.

Abstract. This report presents Milestone 1 of a power system modeling project implemented in Python. The objective of this milestone is to develop foundational equipment classes representing core components of an electric power system, including buses, generators, loads, transformers, and transmission lines. Each class encapsulates key electrical parameters and establishes a structured object-oriented architecture that will be extended in future milestones for power flow analysis and system simulation.

1 Introduction

Modern power system analysis relies heavily on structured software models that accurately represent electrical equipment and network topology. The purpose of Milestone 1 is to establish the foundational equipment classes that will serve as the building blocks for a full power flow solver.

This milestone focuses on creating structured Python classes for the following components:

- Bus
- Generator
- Load
- Transformer
- Transmission Line

Each class defines the necessary electrical parameters as attributes and prepares the overall architecture for integration into a larger solver framework.

2 Data and Methods

This milestone implements object-oriented modeling using Python. Each equipment component is represented as an independent class with defined inputs (constructor parameters) and stored attributes.

2.1 Bus Class

The Bus class represents a node in the power system network. Each bus contains:

- **name** (string): Unique identifier of the bus
- **nominal_kv** (float): Nominal voltage level in kilovolts
- **bus_index** (integer): Automatically assigned index

The bus index is generated using a class-level counter:

```
class Bus :
```

```

index_counter = 1
def __init__(self, name : str, nominal_kv : float) :
    self.name = name
    self.nominal_kv = nominal_kv
    self.bus_index = Bus.index_counter
    Bus.index_counter += 1

```

This ensures that each bus receives a unique index automatically upon instantiation.

```

1 class Bus:
2     index_counter = 1
3
4     def __init__(self, name:str, nominal_kv:float):
5         self.name = name
6         self.nominal_kv = nominal_kv
7         self.bus_index = Bus.index_counter
8         Bus.index_counter += 1
9
10 if __name__ == "__main__":
11     bus1 = Bus( name: "Bus1", nominal_kv: 20.0)
12
13     print(f"Bus1 name: {bus1.name}")
14     print(f"Bus1 Nominal Voltage: {bus1.nominal_kv} Volts")
15     print(f"Bus1 Index: {bus1.bus_index}")
16
17     bus2 = Bus( name: "Bus2", nominal_kv: 230.0)
18
19     print(f"Bus2 name: {bus2.name}")
20     print(f"Bus2 Nominal Voltage: {bus2.nominal_kv} Volts")
21     print(f"Bus2 Index: {bus2.bus_index}")
22

```

2.2 Generator Class

The Generator class models a generator connected to a bus. Its inputs include:

- **name** (string)
- **bus1_name** (string)
- **voltage_setpoint** (float, per-unit)
- **mw_setpoint** (float, MW)

```

class Generator :
    def __init__(self, name : str , bus1_name : str , voltage_setpoint : float ,
                  mw_setpoint : float) :
        self.name = name
        self.bus1_name = bus1_name -
        self.voltage_setpoint = voltage_setpoint
        -
        -

```

```

self.mw_setpoint = mw_setpoint

```

```

1 class Generator:
2     def __init__(self, name:str, bus1_name:str, voltage_setpoint:float, mw_setpoint:float):
3         self.name = name
4         self.bus1_name = bus1_name
5         self.voltage_setpoint = voltage_setpoint
6         self.mw_setpoint = mw_setpoint
7
8 if __name__ == "__main__":
9     gen1 = Generator( name: "G1", bus1_name: "Bus 1", voltage_setpoint: 1.04, mw_setpoint: 100.0)
10
11     print(f"Generator1 Name: {gen1.name}")
12     print(f"Generator1 Bus Name: {gen1.bus1_name}")
13     print(f"Generator1 Voltage Setpoint (V): {gen1.voltage_setpoint}")
14     print(f"Load1 Real Power Setpoint (MW): {gen1.mw_setpoint}")

```

14:27 CRLF UTF-8 4 spaces Python 3.12 (Project 2)

2.3 Load Class

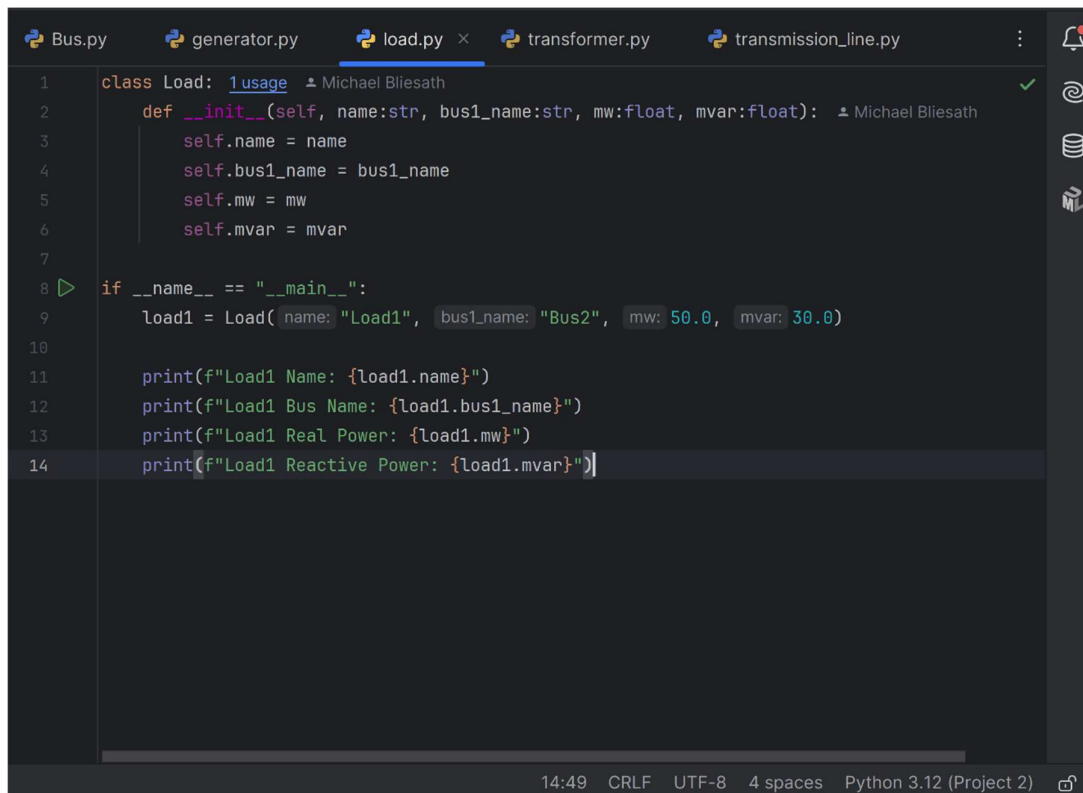
The Load class represents a power demand connected to a bus. Inputs include:

- **name** (string)
- **bus1_name** (string)
- **mw** (float): Real power demand
- **mvar** (float): Reactive power demand

```

class Load :
    def __init__ (self , name : str, bus1_name : str, mw : float, mvar :
        float) :
        self.name = name
        self.bus1_name = bus1_name
        self.mw = mw
        self.mvar = mvar

```

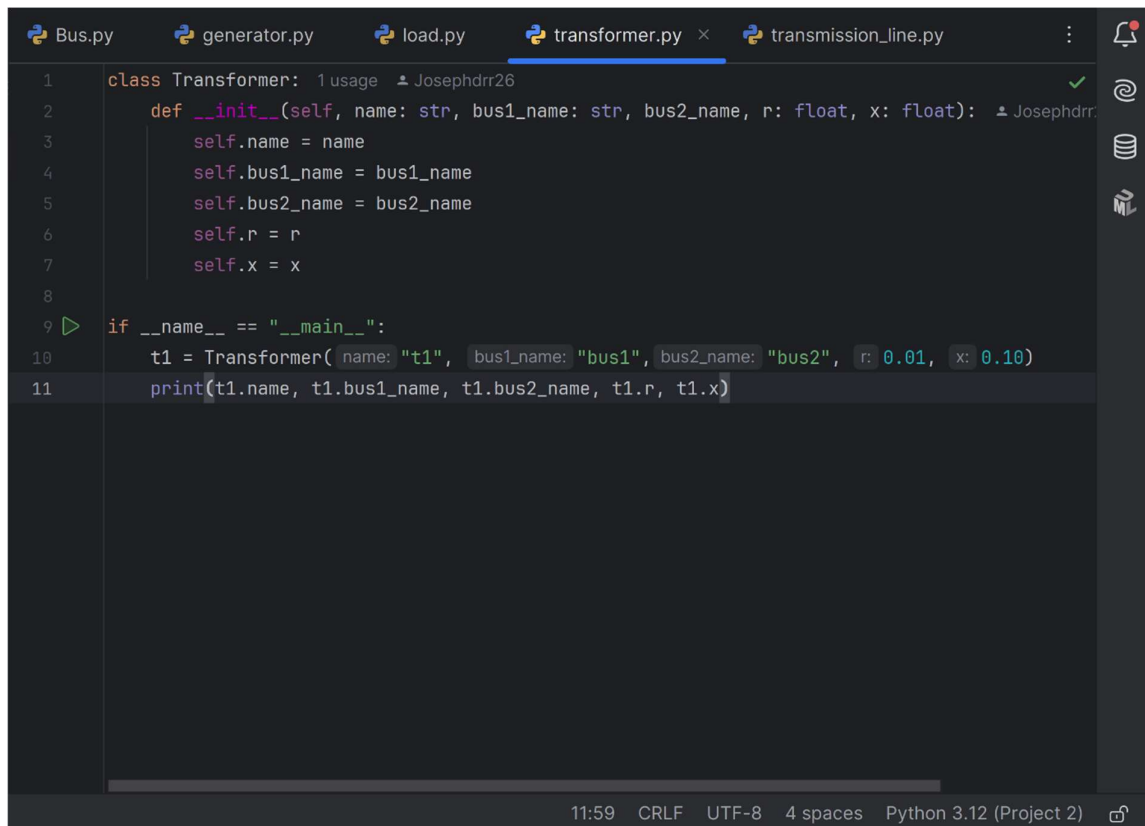
A screenshot of a Python IDE with a dark theme. The top bar shows several open files: Bus.py, generator.py, load.py (selected), transformer.py, and transmission_line.py. The main editor area displays the code for the 'Load' class and its usage. The class 'Load' is defined with an '__init__' method that takes 'name', 'bus1_name', 'mw', and 'mvar' as arguments and assigns them to instance variables. Below the class definition, there is a main block that creates an instance 'load1' with specific values and prints out its attributes. The status bar at the bottom indicates the current time is 14:49, the file encoding is UTF-8, it uses 4 spaces for indentation, and it's running Python 3.12 on a project named 'Project 2'.

```
1 class Load: 1usage  Michael Bliesath
2     def __init__(self, name:str, bus1_name:str, mw:float, mvar:float):  Michael Bliesath
3         self.name = name
4         self.bus1_name = bus1_name
5         self.mw = mw
6         self.mvar = mvar
7
8 if __name__ == "__main__":
9     load1 = Load(name="Load1", bus1_name="Bus2", mw=50.0, mvar=30.0)
10
11     print(f"Load1 Name: {load1.name}")
12     print(f"Load1 Bus Name: {load1.bus1_name}")
13     print(f"Load1 Real Power: {load1.mw}")
14     print(f"Load1 Reactive Power: {load1.mvar}")
```

2.4 Transformer Class

The Transformer class models a transformer connecting two buses. It includes resistance and reactance parameters:

```
class Transformer :
    def __init__(self, name : str, bus1_name : str, bus2_name : str, r
        : float, x : float) :
        self.name = name
        self.bus1_name = bus1_name
        self.bus2_name = bus2_name
        self.r = r
        self.x = x
```

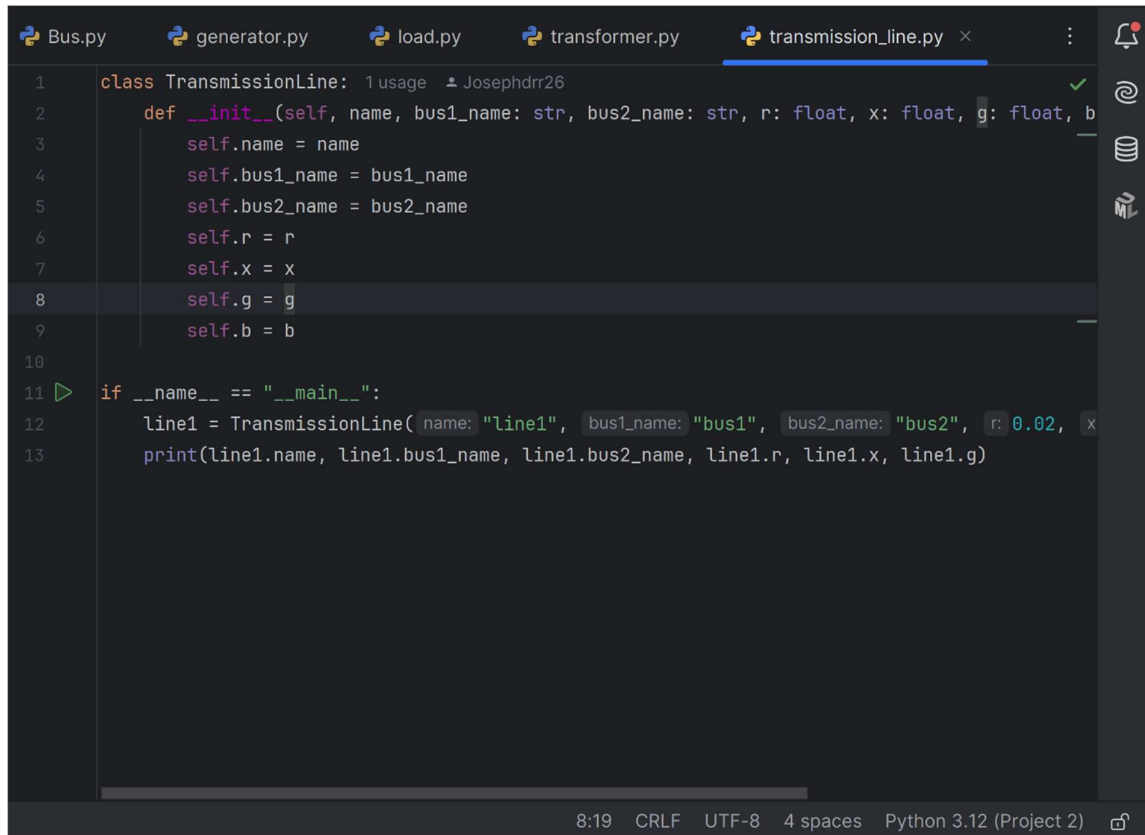


```
1 class Transformer:
2     def __init__(self, name: str, bus1_name: str, bus2_name: str, r: float, x: float):
3         self.name = name
4         self.bus1_name = bus1_name
5         self.bus2_name = bus2_name
6         self.r = r
7         self.x = x
8
9 if __name__ == "__main__":
10     t1 = Transformer(name="t1", bus1_name="bus1", bus2_name="bus2", r=0.01, x=0.10)
11     print(t1.name, t1.bus1_name, t1.bus2_name, t1.r, t1.x)
```

2.5 Transmission Line Class

The TransmissionLine class models a transmission line using series impedance and shunt admittance parameters:

```
class TransmissionLine :
    def __init__(self, name : str, bus1_name : str, bus2_name : str , r
                  : float, x : float, g : float, b : float) :
        self.name = name
        self.bus1_name = bus1_name
        self.bus2_name = bus2_name
        self.r = r
        self.x = x
        self.g = g
        self.b = b
```



```
1 class TransmissionLine:
2     def __init__(self, name, bus1_name: str, bus2_name: str, r: float, x: float, g: float, b: float):
3         self.name = name
4         self.bus1_name = bus1_name
5         self.bus2_name = bus2_name
6         self.r = r
7         self.x = x
8         self.g = g
9         self.b = b
10
11 if __name__ == "__main__":
12     line1 = TransmissionLine(name="line1", bus1_name="bus1", bus2_name="bus2", r=0.02, x=0.1, g=0.0, b=0.0)
13     print(line1.name, line1.bus1_name, line1.bus2_name, line1.r, line1.x, line1.g)
```

3 Results

The implemented classes successfully instantiate equipment objects with proper attribute storage. Each class can be independently tested using a main block, confirming:

- Proper attribute assignment
- Unique bus index generation
- Structured storage of electrical parameters

These validated equipment models establish the structural foundation for subsequent solver development.

4 Conclusion

Milestone 1 establishes the object-oriented architecture required for a power system modeling framework. By implementing core equipment classes, the project now has structured representations of buses, generators, loads, transformers, and transmission lines.

The automatic indexing of buses ensures consistent system organization, while clear parameter definitions prepare the system for integration into future power flow solvers.

Future milestones will integrate these classes into a unified circuit representation and implement numerical solution methods for steady-state power flow analysis.

Conflict of interests

None.

Acknowledgments

Not applicable for Milestone 1.

Funding

Not applicable.

Availability of data and software code

All software code was developed in Python as part of this milestone and is maintained within the project repository.