GridWorkbench User Guide

Adam Birchfield, Texas A&M University, 2/1/2022

GridWorkbench is a Python-based package that holds data structures for power system analysis. It is available for installation with "pip install gridworkbench". The package is still under development and bug reports and feature requests should be sent to abirchfield@tamu.edu.

The GridWorkbench Object

This object is often the only thing that needs to be imported, creating a workbench object to use for the remainder of the program.

```
from gridworkbench import GridWorkbench
wb = GridWorkbench()
```

The **clear** function resets the workbench to an empty state, as it was when created. Any custom or package-specific data may be retained.

```
wb.clear()
```

Container Objects

The system in GridWorkbench is organized into a hierarchy of containers, with each level containing zero or more containers of the lower level. From highest to lowest they are: Region, Area, Sub, Bus, Node. Every container has a unique number in its class.

Region

A region is a very large portion of the system. Most systems except the very largest will only have one region. A region is directly under the workbench, and identified by a number. To access a single region, specify its number.

```
region = wb.region(1)
```

Or, you can iterate through all the regions in the workbench.

```
for region in wb.regions:
    print(region.number)
```

Regions will always have a number and workbench object (region.number and region.wb). Other properties are application-specific and can be added. Regions correspond to PowerWorld's SuperAreas.

Area

An area is a large portion of the system, usually the equivalent of a balancing authority. Smaller systems will typically have only one area. To access an area, use either the region or workbench object and either specify the number or iterate over all values.

```
area = wb.area(4)
area = region.area(4)
for area in wb.areas:
    print(area.number)
for area in region.areas:
    print(area.number)
```

From an area object, the number, region, and workbench are accessible. The third example below prints the total number of areas in the workbench.

```
print(area.number)
print(area.region.number)
print(len(list(area.wb.areas)))
```

Areas can have other properties that are application-specific.

Sub

A sub object represents a substation, the portion of the system that is at a single geographic site. If substation identities are unknown, normal practice would be to assume one bus per substation. Substations can be accessed by number by the containing area or the overall workbench. Subs can be iterated over by the containing area, region, or workbench.

```
sub = wb.sub(4)
sub = area.sub(4)
for sub in wb.subs:
    print(sub.number)
for sub in region.subs:
    print(sub.number)
for sub in area.subs:
    print(sub.number)
```

From a sub object, the number, area, region, and workbench are all accessible.

```
sub.number
sub.wb
sub.region
```

```
sub.area
```
It is very common for subs to have the properties latitude and longitude, but this is not required. Subs may also have other application-specific parameters.

Bus

Buses represent electrical points on the system, within a substation at a single nominal voltage level. Like the other containers, they have access to all enclosing objects and can be accessed by them as well, both from the bus number and by iterator.

```
bus.number
bus.sub
bus.area
bus.region
bus.wb
bus = wb.bus(1)
bus = sub.bus(1)
for bus in wb.buses:
    pass
for bus in region.buses:
    pass
for bus in area.buses:
    pass
for bus in sub.buses:
    pass
```

Buses will typically have the properties nominal_freq and nominal_kv, but this is not required. Buses can also have a zone number, although zones are not container objects in GridWorkbench.

Node

Nodes represent atomic electrical points on the system, which have devices connected to them. Typical modeling will only have one node per bus. However, nodes are useful when detailed modeling, including zero-impedance branches, is intended. All devices are connected to nodes.

```
node.number
node.bus
node.sub
node.area
node.region
node.wb
node = wb.node(1)
node = bus.node(1)
for node in wb.nodes:
```

```
    pass
for node in region.nodes:
    pass
for node in area.nodes:
    pass
for node in sub.nodes:
    pass
for node in bus.nodes:
    pass
```

A node will typically have the technical circuit parameters, like vpu and vang. But that is application-specific.

Device Objects

Devices are the components of the power system. All devices are connected to nodes. They are defined by the node or nodes they connect and a 1- or 2-character string.

Gen

A gen object is a one-node object that represents a generator. It can be accessed from the node or by the workbench if the node number is known.

```
gen = node.gen("1")
gen = wb.gen(584, "1")
gen = wb.node(584).gen("1")
```

You can also iterate through the generators included in any container using the gens keyword.

```
for gen in node.gens:
    pass
for gen in bus.gens:
    pass
for gen in sub.gens:
    pass
for gen in area.gens:
    pass
for gen in region.gens:
    pass
for gen in wb.gens:
    pass
```

Generators have properties for all enclosing containers, their ID, and the workbench.

```
gen.id
gen.node
gen.bus
gen.sub
gen.area
gen.region
gen.wb
```

Load

A load is a one-node object that represents a load. Access is identical to that of generators, with the keywords “load” for finding a single load by node number and id, or “loads” for getting an iterator of all loads in the container.

Shunt

A shunt is a one-node object that represents a shunt device, such as a capacitor or reactor. Access is identical to that of generators, with the keywords “shunt” for finding a single shunt by node number and id, or “shunts” for getting an iterator of all shunts in the container.

Branch

A branch is a generic, passive two-node object, such as a transmission line, transformer, or breaker. The two nodes are specified as from node and to node. The branch can be accessed from either node or from the workbench:

```
branch = node1.branch_from(2, "1")
branch = node2.branch_to(1, "1")
branch = wb.branch(1, 2, "1")
```

It is possible to iterate through branches for any container. For bus and node containers, it is also possible to get only those branches which have the from or to sides in the enclosing container.

```
for branch in wb.branches:
    pass
for branch in region.branches:
    pass
for branch in area.branches:
    pass
for branch in sub.branches:
    pass
for branch in bus.branches:
    pass
for branch in bus.branches_from:
```

```
    pass
for branch in bus.branches_to:
    pass
for branch in node.branches:
    pass
for branch in node.branches_from:
    pass
for branch in node.branches_to:
    pass
```

Branches have access to nodes, buses, and the workbench. Enclosing subs, areas, and regions should be found by the associated node.

```
branch.from_node
branch.to_node
branch.from_bus
branch.to_bus
branch.wb
branch.from_node.area
branch.to_bus.region
branch.from_bus.sub
```

Converter

Not yet implemented – it is expected to implement an active two-node device to represent power electronics based converters and other active devices.

ThreeWinder

Not yet implemented – this will be a four node device that can model a three-winding transformer (three windings and one star node).

Building A Model with GridWorkbench

After creating the wb object as described above, you need to add data to it. There are several ways to do this.

Adding Containers and Devices Individually

To add devices individually, you need to import the associated classes:

```
from gridworkbench.containers import Region, Area, Sub, Bus, Node
from gridworkbench.devices import Gen, Load, Shunt, Branch
```

Then you can create the objects with the following constructors. Creating the objects is all that is necessary. The package will ensure the objects are properly linked. For containers, the enclosing container must exist prior to creating it (except regions). For devices, the associated nodes must already exist when they are created. Each container number must be unique on the workbench. Each device id must be unique for that set of nodes.

```
region = Region(wb, 1)
area = Area(region, 1)
sub = Sub(area, 1)
bus1 = Bus(sub, 1)
bus2 = Bus(sub, 2)
node1 = Node(bus1, 1)
node2 = Node(bus2, 1)
gen = Gen(node1, "1")
load = Load(node2, "1")
shunt = Shunt(node2, "1")
branch = Branch(node1, node2, "1")
```

Importing from PowerWorld using Easy SimAuto

GridWorkbench uses Easy SimAuto to bring in data from PowerWorld, if SimAuto is installed. Here's an example of how to do this:

```
wb = GridWorkbench()
fname = r"C:\path\...\wscc9.pwb"
wb.open_pwb(fname)
wb.pwb_read_all()
```

The last function can be rerun to grab data from PowerWorld and put it in the GridWorkbench objects. Similarly, the data can be written to PowerWorld using the following function.

```
wb.pwb_write_all()
```

The Easy SimAuto object can be accessed via wb.esa once wb.open_pwb has been run. For example, to save and close the file, do

```
wb.esa.SaveCase(fname)
wb.close_pwb()
```

The exact fields that are read and written to ESA using pwb_read_all and pwb_write_all can be customized by the workbench fields: sa_pw_fields, area_pw_fields, sub_pw_fields, bus_pw_fields, gen_pw_fields, load_pw_fields, shunt_pw_fields, branch_pw_fields. These already have some parameters to start, but more can be added. To add a field, just append a

length-four tuple to the list with values (1) the name you want in GridWorkbench, (2) the PowerWorld field name, (3) a default value, (4) a function to convert the data when it is read. For example, to add the field Zone Number to the bus object:

```
wb.bus_pw_fields.append(("zone_number", "ZoneNum", 0, pw2py_default))
```

Now when wb.pwb_read_all or wb_pwb_write_all is written, this field will be read and written to PowerWorld. Fields can be removed in a similar way by editing these lists.

Saving and Opening JSON Files

GridWorkbench has a custom JSON format that can be used to efficiently save and open cases to save time. Just us the following code:

```
wb.json_open("Test.json")
wb.json_save("Test2.json")
```