Cheatography

PyIPSA Cheat Sheet

by ipsa_power via cheatography.com/159139/cs/33570/

About PyIPSA

PyIPSA is the scripting application programming interface (API) designed to give engineers with comfortability in Python the chance to build more sophisticated network models. The entire code is built on the principles of object oriented programming, courtesy of our very accessible and powerful PyBind wrapper. The cheat sheet below gives scripters a solid foundation to start their IPSA experiences and give their computers more automated control over network design and functionality.

Starting PyIPSA

Starting PyIPSA is as easy as making an interface and uploading the network from there, within a Python console or script

Running Python within IPSA

```
import ipsa
isci =
ipsa.GetScriptInterface()
inet = isci.G etN etw -
ork()
```

Running from console /IDE

```
import ipsa
isci =
ipsa.IscInterface()
fname =
"some_network.i2f"
inet = isci.R ead Fil -
e(f name)
```

This is the first level of building a network within IPSA. From this step, you can use the full API functions to modify the network and run many study types.

Creating a Network

Building a network from scratch can seem daunting but this is done in a similar way to the IPSA UI.
First you designate any busbars that you need and build it up from there

```
new_net = isci.C rea teN ewN etw ork (10 0,50,
True, True, 0 0)
buses = [None] * 5
branches = []
for i in range(5):
  bus[i]=new_net.CreateBusbar("Bus "+str(i))
bid=0
for sid in [bus.G etUid() for bus in buses]:
  for rid in [bus.G etUID() in buses != sid]:
    branches[bid]=new_net.CreateBranch(sid, rid,
str(bid))
    bid+=1
```

Tip: It's a good rule of thumb to make all the busbars first and then build from there

Accessing Line Information

List of Components in PyIPSA

Users can script all our IPSA components within their PyIPSA networks

Name	Python Code
Busbar	`IscBusbar`
Branch	`IscBranch`
Two Winding Transformer	`IscTra nsf ormer`
Three Winding Transformer	`Isc3WT ran sformer`
Load	`IscLoad`
Induction Motor	`IscInd Machine`
Synchronous Generator	`IscSyn Machine`
Grid Infeed	`IscGri dInfeed`
Harmonic Source	`IscHar monic`
Universal Machine	`IscUMa chine`

For more components in IPSA check out our PyIPSA ReadTheDocs

Redrawing Networks in Python

The IPSA UI allows you to graphically modify your drawn networks but PyIPSA gives you the chance to automate this

```
idgr, ix = inet.GetAllDiagrams(),1
for nUid in [bus.G etUID() for bus in buses]:
  idgr[0].DrawBusbarCircular(nUid,20,ix,ix)
  ix += 1
  idgr[0].DrawUndrawnItemsAttachedToBusbar(nUid)
```

Components and Access Functions

Two-winding transformers in IPSA are branches with tap-changers mounted ontop. In this case, you need to edit the specific branch information with unique functions such as `GetILi neValue`

```
tf_maxtap =
tx.GetDValue(ipsa.IscTransformer.MaxTapPC)
tf_resistance = tx.Get Lin eDV alu e(i psa.Is cBr -
anc h.R esi stance)
```

For example, while the tap variables are targeted using `IscTra - nsf ormer`, the impedance values are targeted using `IscBranch`.

Note that the same applies for the Set functions above as well

Every component in IPSA has an associated class which can be added, modified or destroyed from your network. These all share the same access functions that require the user to input a particular field value reference (which the code takes as an integer).

```
bus1 = inet.GetBusbar(1)
bus1_voltage = bus1.G etD Val ue( ips a.I scB usb -
ar.N om VoltkV)
```

This also works for strings, integers and booleans:

```
b1_name = bus1.GetSValue(ipsa.IscBusbar.Name)
b1_ctrl = bus1.G etI Val ue( ips a.I scB usb ar.C -
on tro lType)
```

Plus we can set values in a similar way:

```
bus_volt = 33.
bus1.SetDValue(ipsa.IscBusbar.NomVoltkV,bus_volt)
```

Tip: You can also access all the network elements using dictionaries and the `iNet.G etB usb ars` syntax, where the keys are the element names!

Cheatography

PyIPSA Cheat Sheet

by ipsa_power via cheatography.com/159139/cs/33570/

Running an Analysis Study

The core principle of IPSA is to run analysis modules such as load flow and fault level in order to evaluate network feasibility and capacity (among many other functions). Once you have built your network, you can specify the run settings in the `IscAna lysisLF` class and run the `DoLoad Flow()`function as shown below.

```
lfset = inet.GetAnalysisLF()
lfset.SetIValue(ipsa.IscAnalysisLF.LockTaps) = 1
# To use minimum resistance value in multi- -
section lines
lfset.SetIValue(ipsa.IscAnalysis.WhichImpedance = 1
inet.DoLoadFlow()
```

Important: This is slightly more complicated for a harmonics or fault analysis. For example, in a harmonic analysis, you have to specify each of the impedance coefficients for lines, transformers etc and also all the specific variables within each of the components.

Computing Load Profiles

To run a series of load flow scenarios designed for generators and loads you need to build the scenario set first using dictionaries

```
cats = {0:'PF1', 1:'PF2', 2:'PF3'}
apower = {0:0.8, 1:0.775, 2:0.75}
rpower = {0:0.48, 1:0.465, 2:0.45}
```

Then we build a profile instance defined by the class `IscLoa dPr - ofi leP QActual` using the `IscNetwork` function

```
profUID = inet.CreateLoadProfilePQActual('test')
prof = inet.GetLoadProfilePQActual('test')
prof.SetCategoryNames(cats}
prof.SetPMW(apower)
prof.SetQMVAr(rpower)
```

Finally you just have to attach this profile to the load in question

```
load =
ipsa_net.CreateLoad(send.GetUID(),rec.GetUID)
load.SetIValue(ipsa.IscLoad.ProfileUID,profUID)
```

To run this correctly, make sure that you have `ProfileUse = 1` and iterate through the `Profil eLo adC - ategory` value that refers to the strings declared in `IscLoa dPr ofi leP QAc tua l.S etC ategory`

Finishing Touches

When you have finished working on your network, don't forget to use the functions to save the file you've been working on. Otherwise you will lose your progress:

```
inet.WriteFile("C:\Documents\new_network.i2f
bClosedOK = isci.C los eNe twork()
```

Note: Try not to open, run or save any IPSA networks if you have them open in the IPSA UI program as well. PyIPSA can only open a file once at a time, same way as the IPSA UI will.

Additional Packages

When using PyIPSA you might find that some additional packages make analysis easier:

`numpy`	`scipy`	`opencv`
`pandas`	`matplotlib`	`seaborn`
`openpyxl`	`numba`	`setuptools`

There are many more useful libraries but these are the ones that we know users utilise with PyIPSA