OpenDSS Training Workshop

Interfaces in OpenDSS

Davis Montenegro Ph.D.
Technical leader

Web – OpenDSS training
08/31/2021

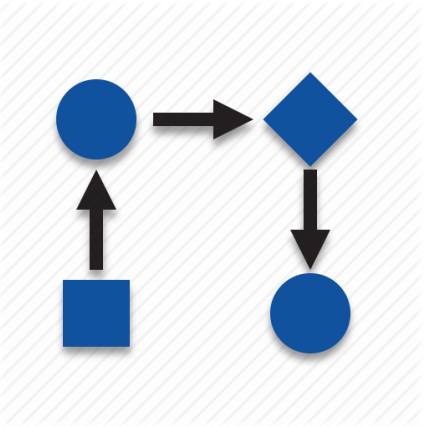
Instructor

Davis Montenegro, *Senior Member, IEEE*

Davis Montenegro-Martinez serves as technical leader at the Electric Power Research Institute (EPRI) in the areas of power system modeling, analysis and high-performance computing. He received his degree in electronics engineering from Universidad Santo Tomás, Bogotá, Colombia (2004); he is M.Sc. in electrical engineering from Universidad de los Andes, Bogotá , Colombia (2012). He received his Ph.D. in electrical engineering from Universidad de los Andes (2015), and a Ph.D. in electrical engineering from the University Grenoble-Alpes, France (2015).

Before joining EPRI, Davis served for 10 years as a lecturer for Universidad Santo Tomas in Colombia, during this time he was also technology consultant in the areas of industrial automation, software and electronic hardware design focused in the electric power industry, specifically in monitoring and control for meter calibration laboratories. His expertise in parallel computing techniques is being used at EPRI for incorporating multi-core processing to power system analysis methods such as QSTS, reducing the computational time required to perform these analysis using standard computing architectures

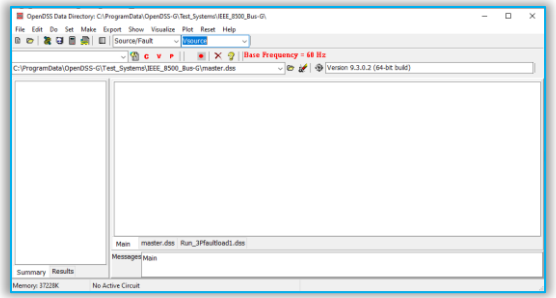
The evolution of OpenDSS into a parallel computing machine



After being released in 2008 as open-source software OpenDSS has become widely used around the world. One of the features that makes OpenDSS popular is that the package offers interfaces for co-simulation.

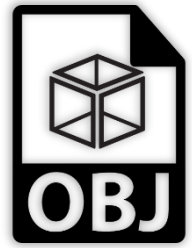
Interfacing with OpenDSS

User Interfaces



» A **stand-alone executable** program that provides a text-based interface (multiple windows)

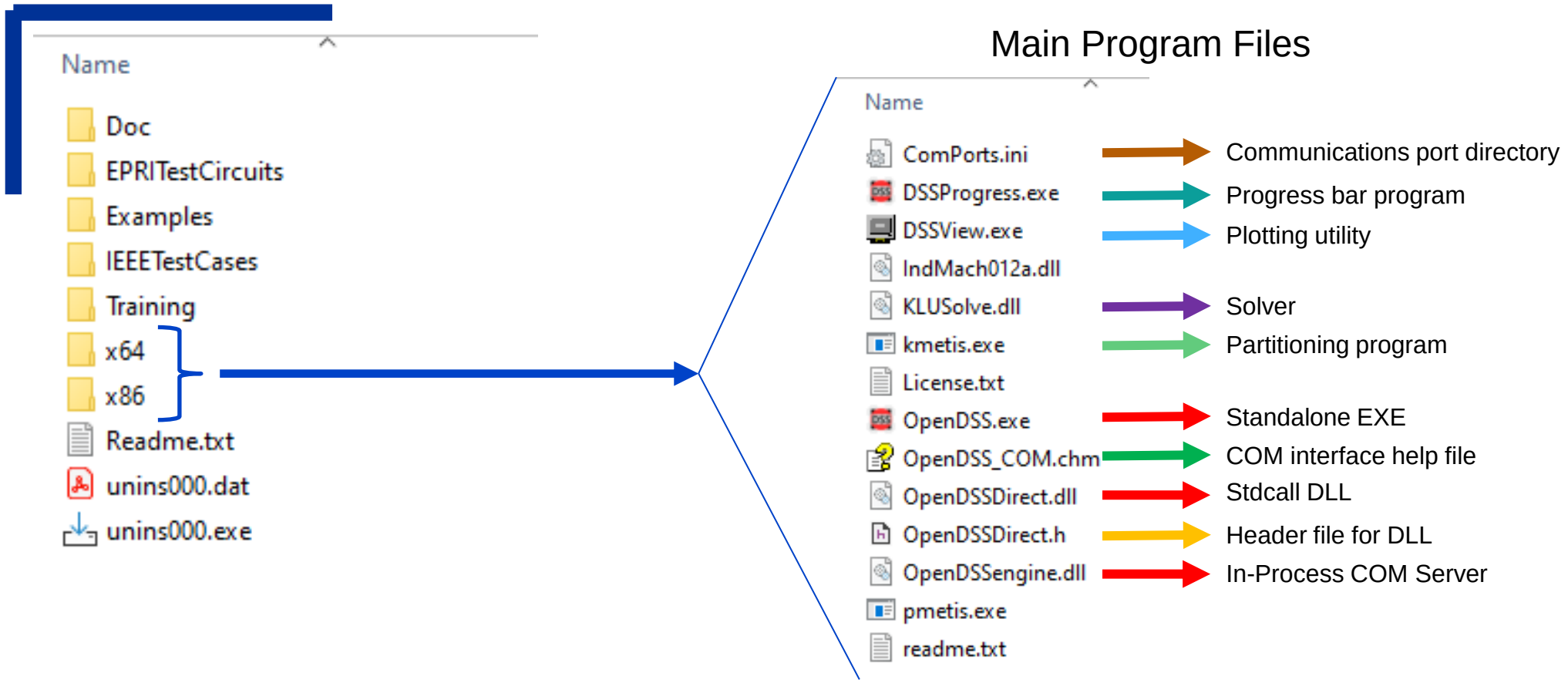
» An **in-process COM server** (for Windows) that supports driving the simulator from user-written programs.



» A **direct DLL** interface that mimics the COM interface

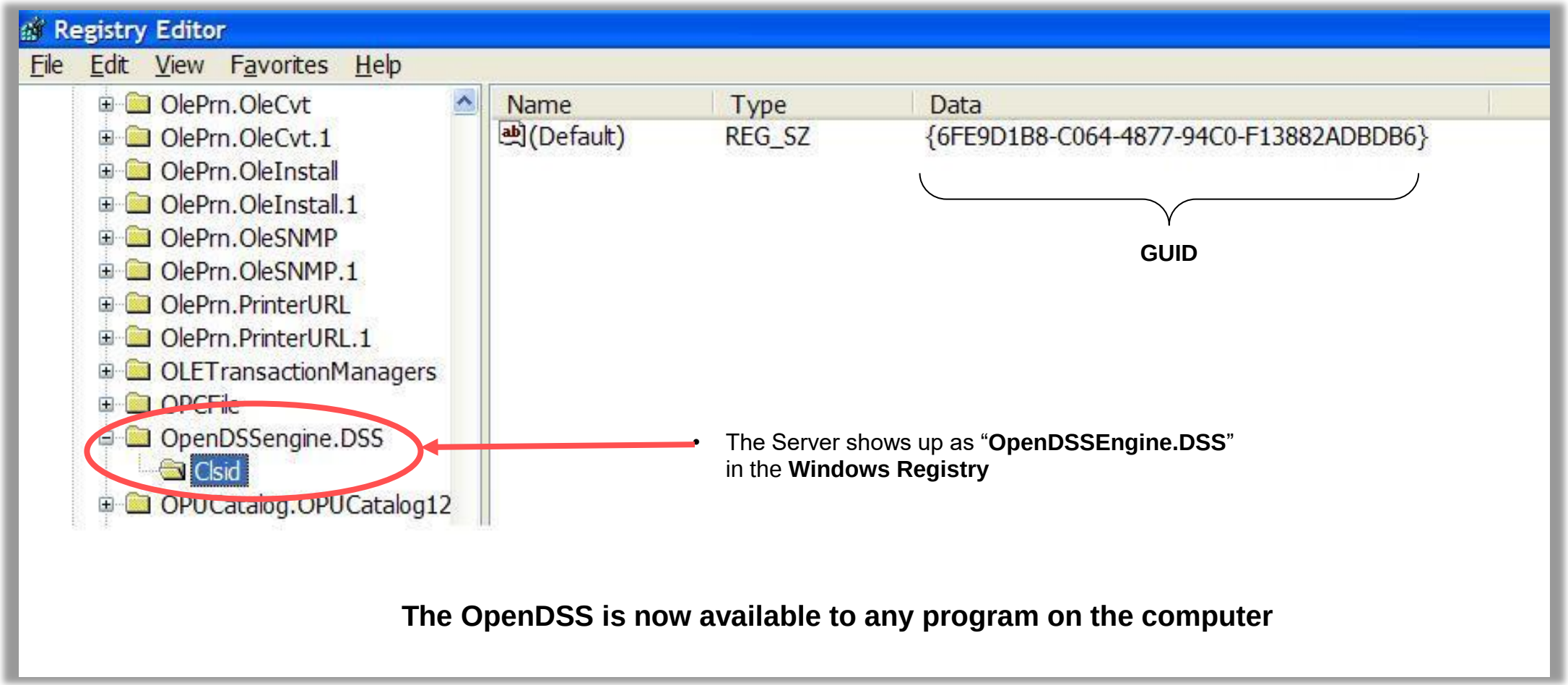
- For non-Windows platforms, such as HPCs
- For programming languages that do not support COM or are not efficient at supporting COM

OpenDSS Files Installed



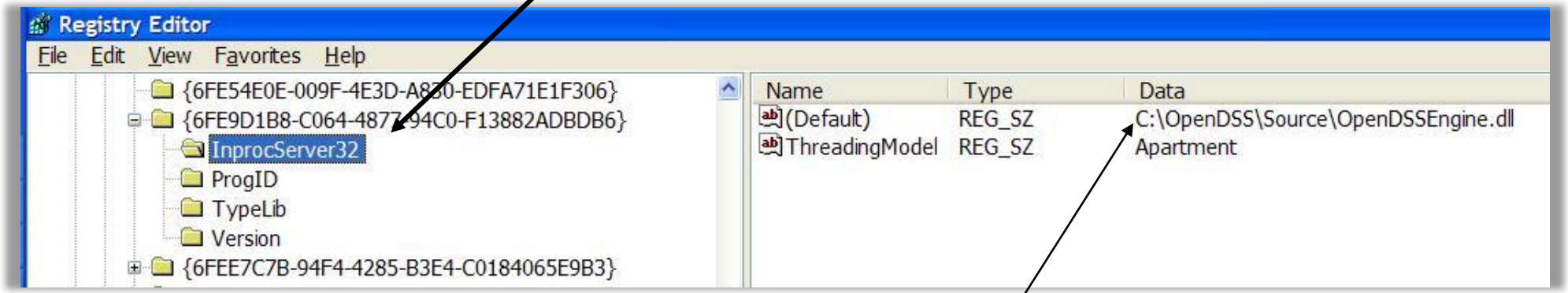
Registering the COM server

Windows Registry Entry



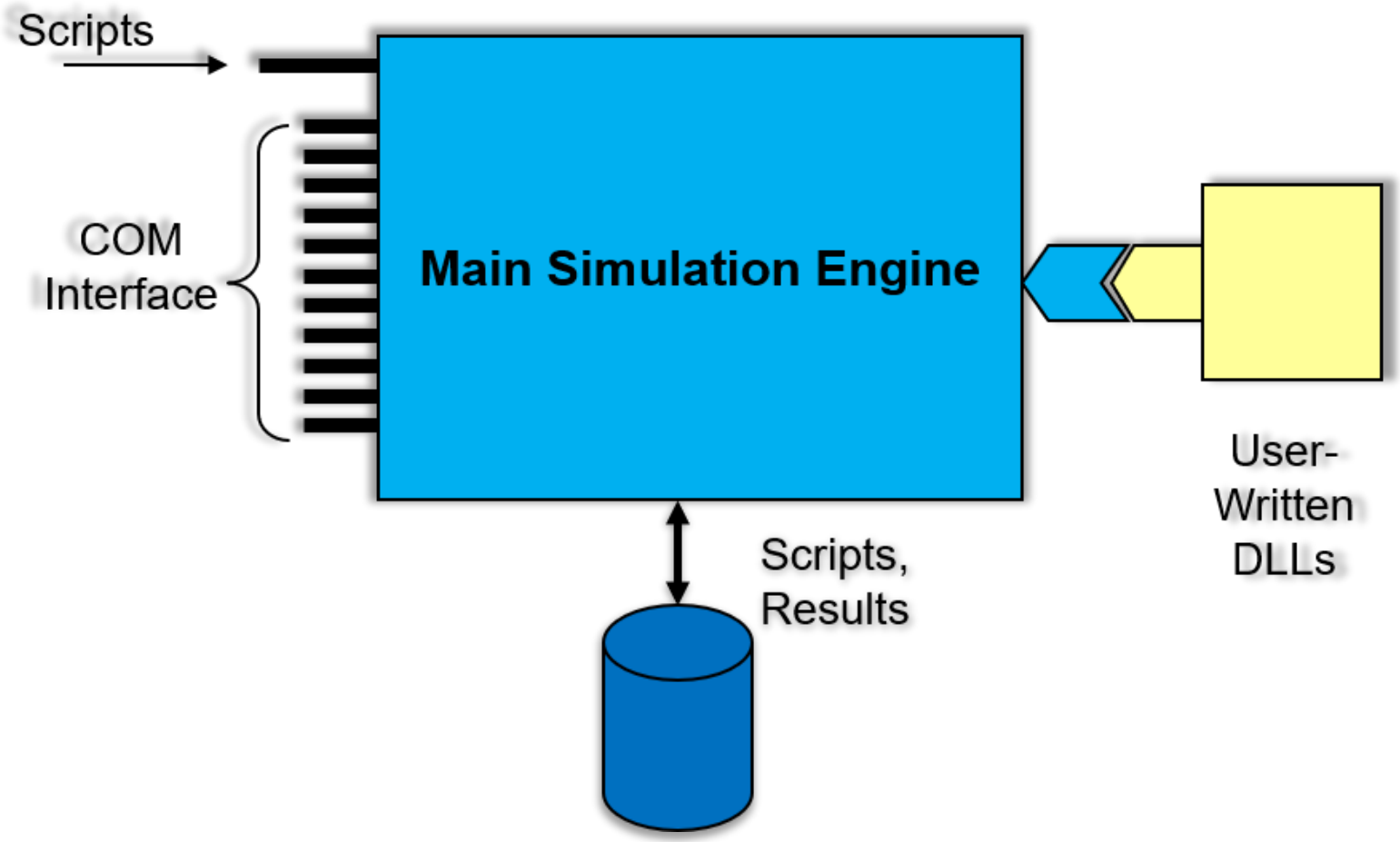
The GUID References the DLL File

If you look up the GUID in RegEdit

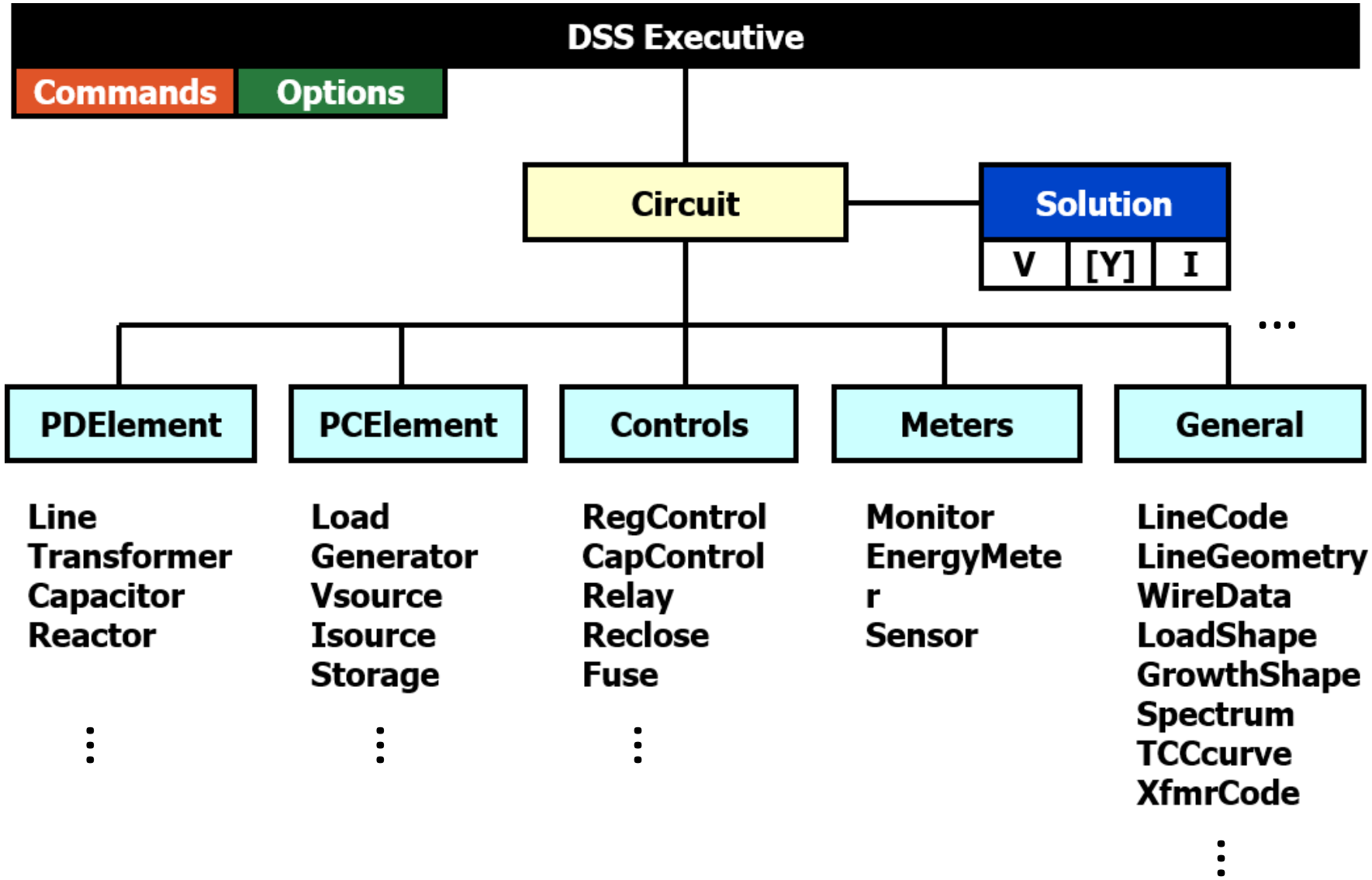


Points to OpenDSSEngine.DLL
(In-process server, Apartment Threading model)

DSS Structure

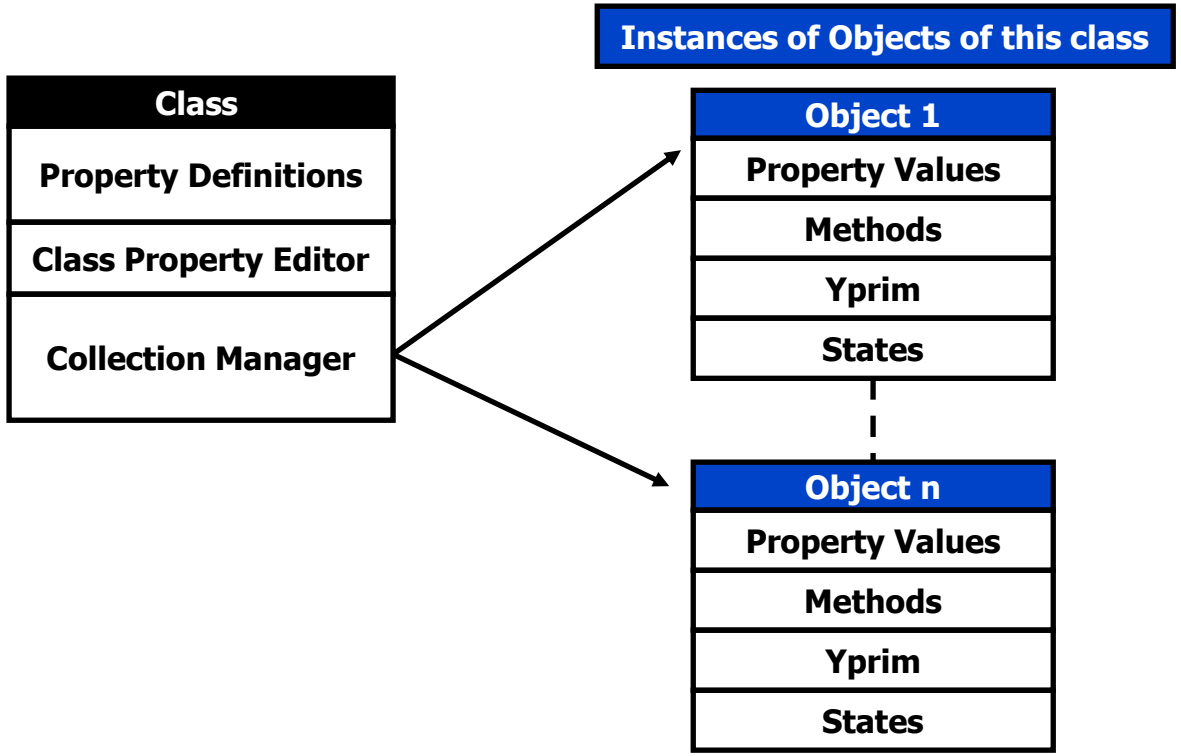


DSS Object Structure



Constantly growing

DSS Class Structure



- Each object has properties and methods
- Each property returns data according to the call type

How can I query about the available interfaces, properties and methods

User Interfaces

- Using a program language that depicts the interface for you (e.g. MS excel, VBA, etc.).

- Reading the documentation: https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Doc/

- Using the query tools available in your programming language:

  MATLAB : get, properties…

  Python: getattr, getAllAttributeNames…

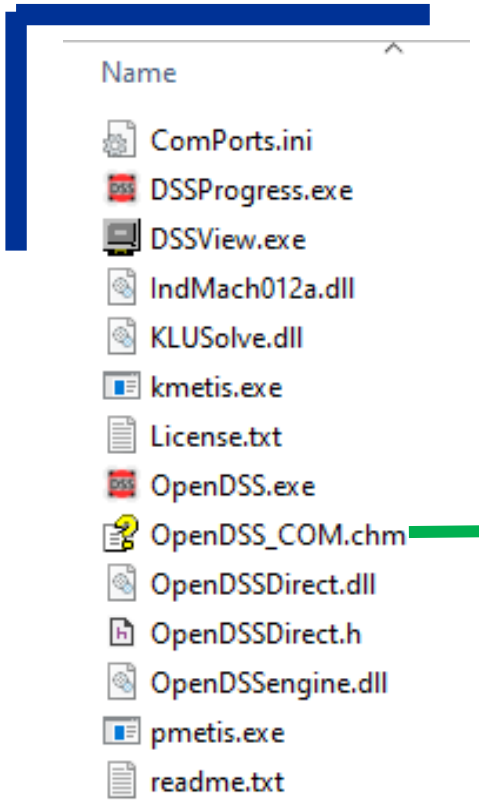
- If working with DirectDLL, then, you'll have to read the documentation and probably use the header file provided.

User Interfaces

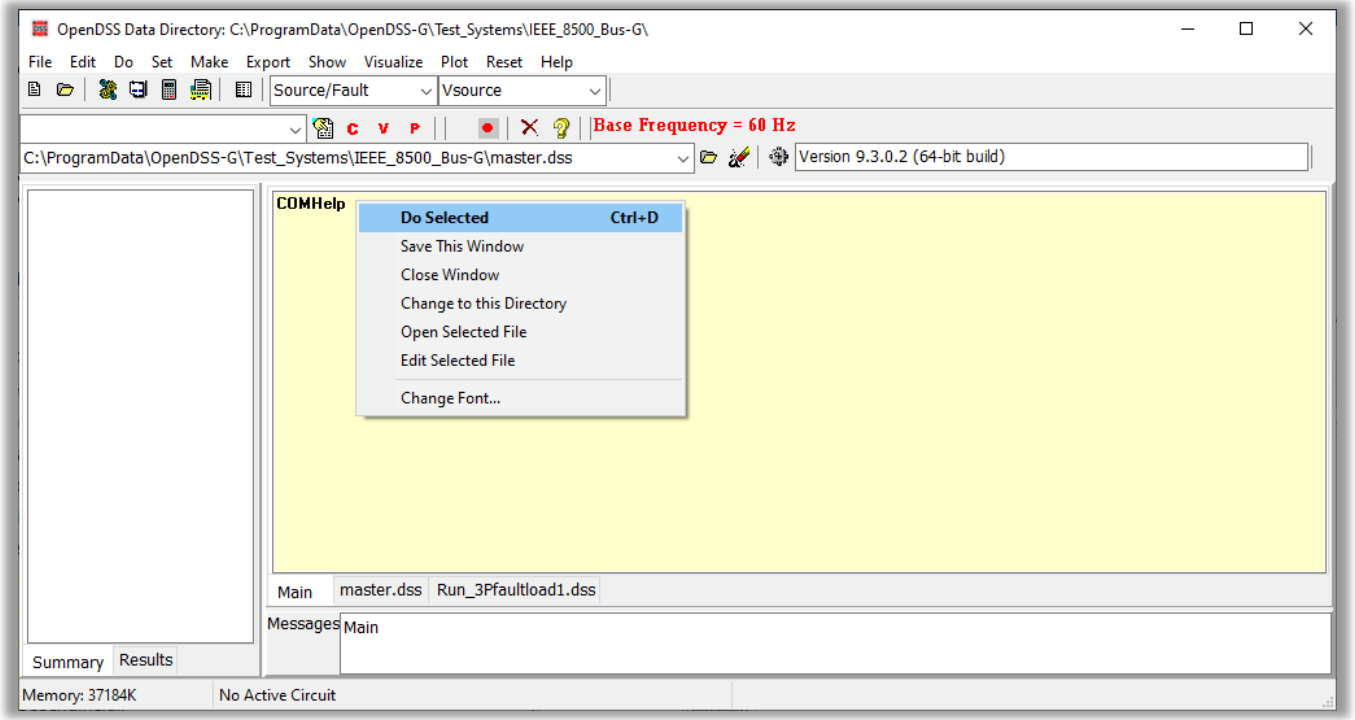
Or…

User Interfaces

» Use the COM help file

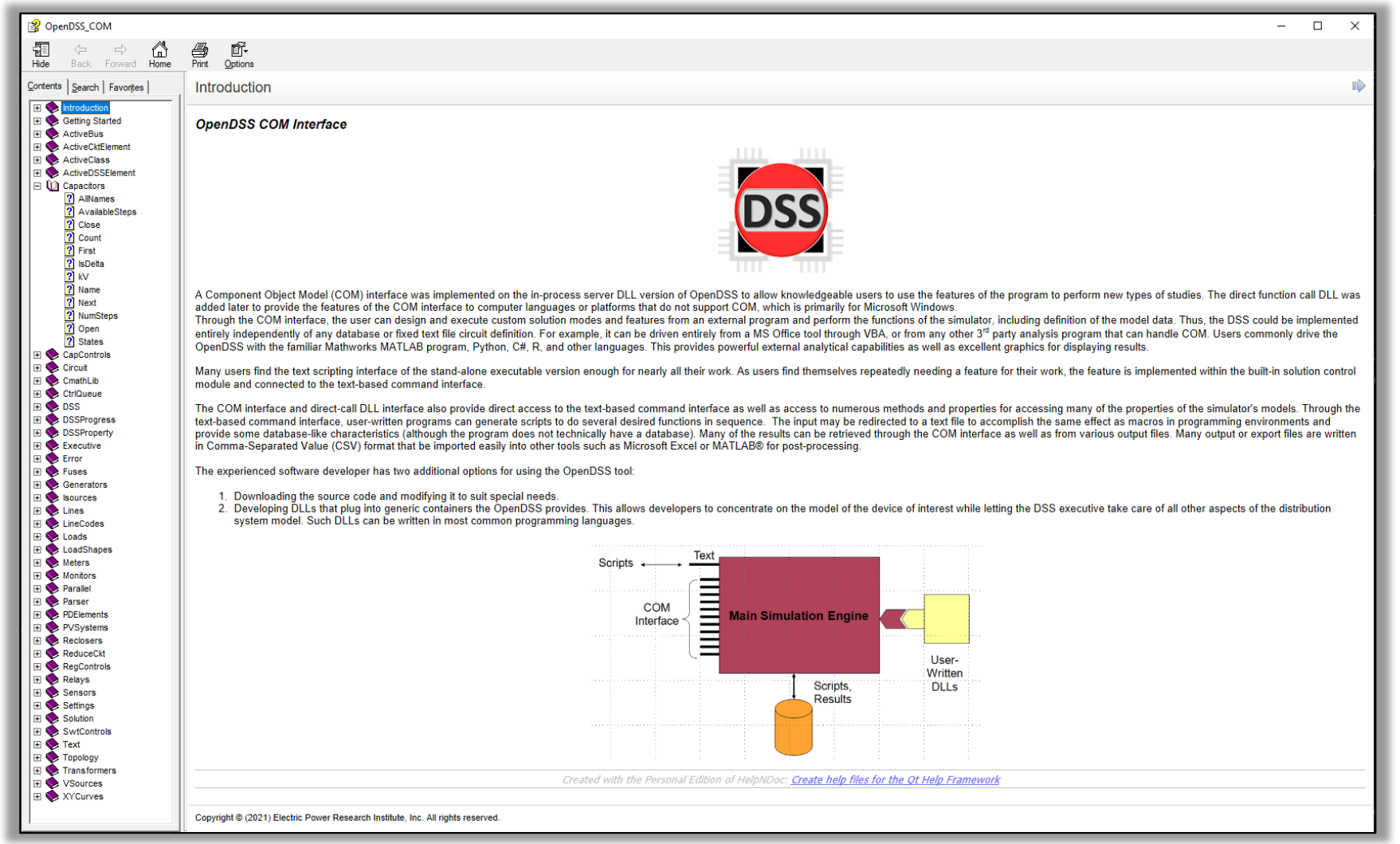


COM interface help file



User Interfaces

» Use the COM help file



User Interfaces

» Use the COM help file

Examples on using COM

Direct DLL (Shared library)

EPRI ELECTRIC POWER RESEARCH INSTITUTE

Direct connection Shared Library (DLL) for OpenDSS

Davis Montenegro, Celso Rocha, Paulo Radatz

Last update 08-26-2021

The direct connection shared library is a DLL that implements the same classes, properties, and methods of the OpenDSS-PM COM interface. This alternative was generated to accelerate the In-process co-simulation between OpenDSS and external software when the client software does not support early bindings connection to COM servers/controls.

Normally, high level programming languages do not support early bindings, which make them use late bindings for data exchanging with COM servers. Late bindings procedures add an important overhead to the co-simulation process specially when executing loops.

So, if your programming language does not support early bindings connection with COM servers, this is the library you should use to accelerate your simulations. This library is called OpenDSSDDirect.dll and can be accessed directly without needing to register it into the OS registry.

However, if your programming language supports early bindings, keep using the COM interface, the simulation speed will be accelerated naturally when using this connection procedure instead of late bindings.

The properties implemented in this library are the same implemented in the COM interface, so, this manual can be used as a reference manual for the classes, properties and methods included in the COM interface.

ActiveClass Interface

This interface implements the ActiveClass (IActiveClass) interface of OpenDSS by declaring 4 procedures for accessing the different properties included in this interface.

ActiveClassI (Int) Interface

This interface can be used to read/modify the properties of the ActiveClass Class where the values are integers. The structure of the interface is as follows:

```
int32_t ActiveClassI(int32_t Parameter, int32_t argument) ;
```

This interface returns an integer (signed 32 bits), the variable "parameter" is used to specify the property of the class to be used and the variable "argument" can be used to modify the value of the property when necessary. Reading and writing properties are separated and require a different parameter number to be executed.

The properties (parameter) are integer numbers and are described as follows:

EPRI ELECTRIC POWER RESEARCH INSTITUTE

Alternative Interfaces

These interfaces were made to cover particular requests and can be available in future versions of this DLL. The structure of these interfaces can be different from the rest of the interfaces proposed above.

InitAndGetYparams Interface

This interface initializes the YMatrix of the system in case of being necessary. It must be executed before executing the interface *GetCompressedYMatrix* to reduce the possibility of errors. The structure of this interface is as follows:

```
Uint32 InitAndGetYparams(uint32 hY, uint32 nBus, uint32 nNZ);
```

GetCompressedYMatrix Interface

This interface gets the YMatrix of the system in a reduced format by delivering the pointers of row, column and matrix values. The structure of this interface is as follows:

```
Void GetCompressedYMatrix(uint32 hY, uint32 nBus, uint32 nNz, uint32 ColPtr, uint32 RowIdx,
                          uint32 cVals);
```

ZeroInjCurr Interface

This interface commands to OpenDSS to initialize currents vector as an array of zeros, which is basically the first step in the solution algorithm (DoNormalSolution routine). The structure of this interface is as follows:

```
Void ZeroInjCurr();
```

GetSourceInjCurrents Interface

This interface commands to OpenDSS to include the currents injected by voltage/current sources into the currents vector, following the procedure proposed in the solution algorithm (DoNormalSolution routine). The structure of this interface is as follows:

```
Void GetSourceInjCurrents();
```

GetPCInjCurr Interface

Direct connection Shared Library (DLL) for OpenDSS

Examples on using Direct DLL

Myths and legends about interfaces

Myths and legends about user Interfaces

» COM interface is slow…

How to speed up your co-simulation using OpenDSS COM interface

Authors: Davis Montenegro, Roger Dugan
October 27, 2015

Many times users have mentioned performance issues when co-simulating using OpenDSS. The interface proposed in OpenDSS to perform co-simulations is the COM interface, which can be accessed from almost every programming language; however, not necessarily in the faster way.

Because of this, many users blame the COM interface for their performance issues. The aim of this document is to clarify why users could experience a low performance in terms of co-simulation speed and to give a guide about how to improve it.

Early Binding and Late Binding

Early and Late bindings are two computer programming mechanisms for accessing COM servers/ActiveX controls considering the features of the interface objects and classes. Late binding is focused on giving flexibility in case the objects/classes contained within the interface are polymorphic. For doing this, the late binding mechanism uses a Virtual table (vtable) that includes the memory address of each allocated class and its features. Every time the external program accesses an object/class using the COM interface it must go to the vtable first (if something changes it will be updated into the vtable), then look for the object to obtain the memory address and its features, which adds significant overhead on each iteration.

In contrast, early binding considers that every object is static in time and will be the same during the connection with the external software. For doing this, the vtable is eliminated and the index to each object/class is made by specifying the DispID memory address directly during the compilation phase. This eliminates the overhead generated by accessing vtables and the access to the COM interface objects is drastically faster [1].

To get connected to the COM interface using early binding, first check to see if your programming language supports early binding connection. Normally, all programming languages support late bindings but not necessarily early binding. However, in the next sections, this article presents an alternative for programming languages that do not support early binding.

If your programming software language supports early binding the activation of this connection mechanism consists of adding a couple of code lines when establishing the connection with the COM server. To evaluate the performance of the early binding vs. late

And remember:

It all depends!

R. C. Dugan

..false

Myths and legends about user Interfaces

» Direct DLL is faster than COM interface…

The Delphi Test Routine

The actual Delphi code for the loop in Figure 1 is:

```
procedure TForm1.Button2Click(Sender: TObject);
Var
     i:Integer;
     iLine : Integer;
     DSSLoads : ILoads;

begin
      SW := TStopWatch.Create();
      DSSLoads := DSSCircuit.Loads ;
      SW.Start;
      With DSSLoads  Do
        for i := 1 to 1000 do  Begin
          iLine := First;
          while iline > 0 do  Begin
               kW := 50.0;
               kvar := 20.0;
               iLine := Next;
          End;
      End;

      SW.Stop ;
      Edit1.Text := Format('%d ms for %d Loads 1000 times',
                           [SW.ElapsedMilliseconds, DSSLoads.count]);
end;
```

The early-binding connection to the OpenDSS COM interface is made using the following code to make a connection to the IDSS interface. Then some local variables are used to achieve a little better performance by maintaining a static connection to other interfaces that are created by the call to "coDSS.Create".

Definition of Variables

```
Var
    DSSObject: IDSS;  // DSS Interface
    DSSText:   IText;
    DSSCircuit: ICircuit;
    DSSSolution:ISolution;
    DSSProgressfrm:IDSSProgress;
```

Connection to the Interface

```
        TRY

            DSSObject := coDSS.Create; // Creates early binding
            DSSObject.Start(0);
        EXCEPT
            On E:Exception Do Begin
              MessageDlg('Did not Start.',  mtConfirmation, [mbYes, mbNo],
0, mbYes);
              Exit;
            End;

        END;
```

VBA Early and Late Binding

VBA (Visual Basic) is a programming language that supports both early binding in a simple way as well as the traditional initialization of the COM interface using late binding.

Late Binding

```
Public DSSObj as Object
....
Set DSSObj = CreateObject("OpenDSSEngine.DSS")
```

Early Binding

```
Public DSSObj as OpenDSSEngine.DSS
...
Set DSSObj = New OpenDSSEngine.DSS
```

The measured computing times using each technique are shown in Figure 2.

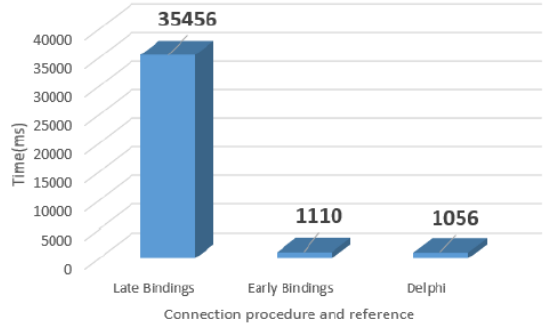


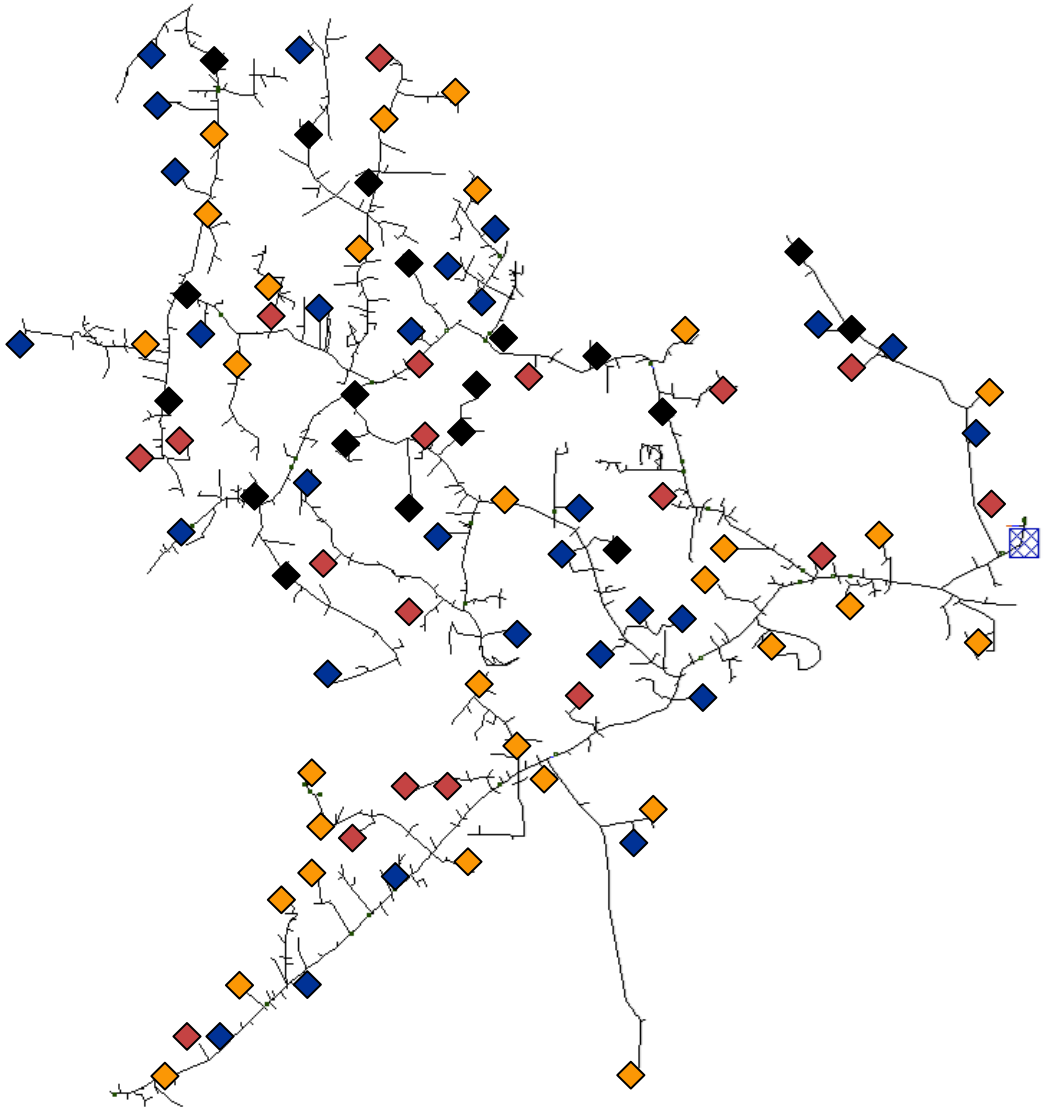
Figure 2. Computing times in *ms* for the algorithm in Figure 1 using VBA late binding and early binding in comparison with the reference (Delphi)

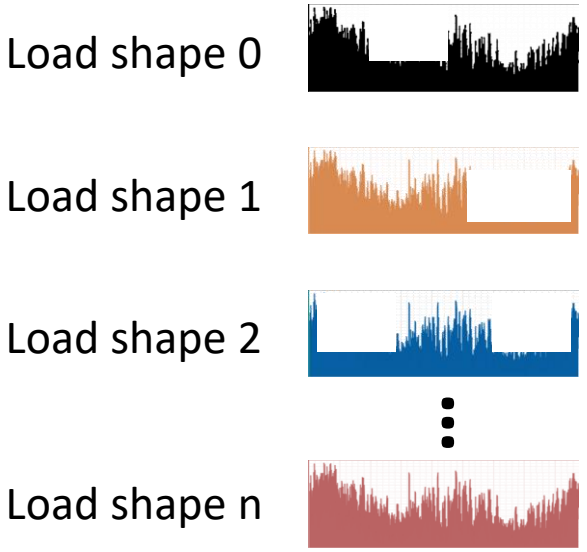
..false

New functionalities in DSS

Load shape aggregation

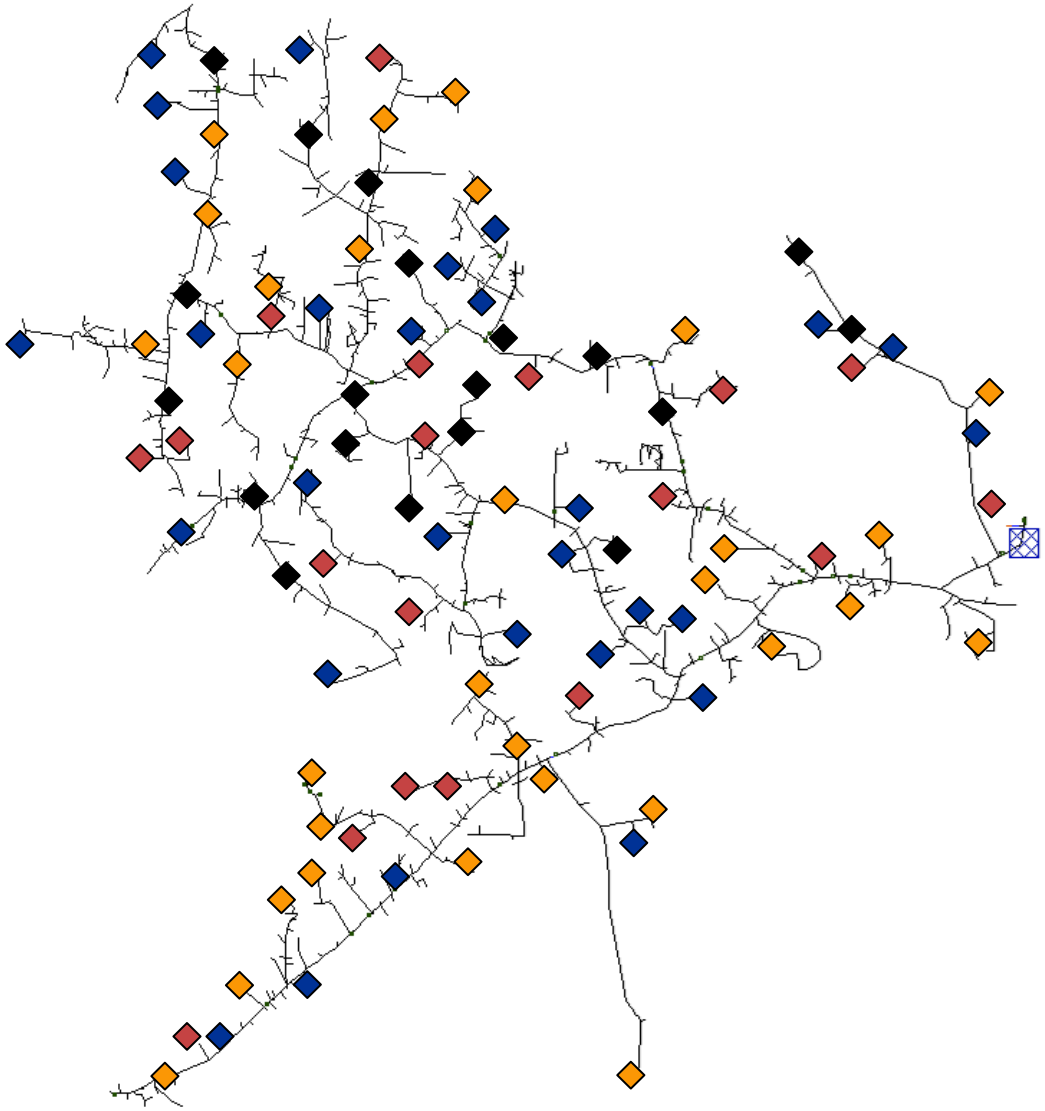
Data diversity within a distribution model

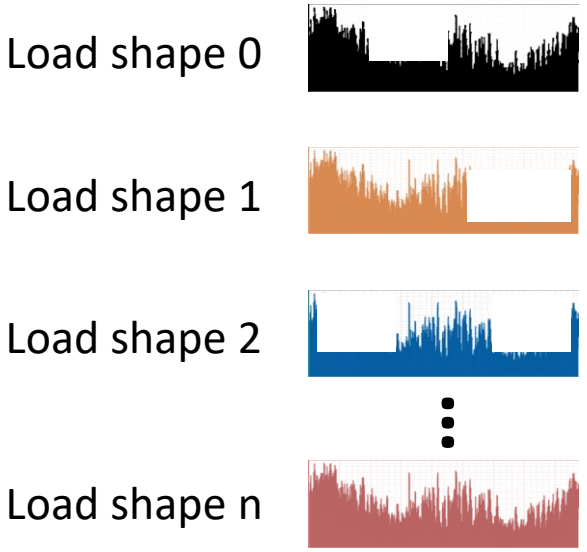




- Abundant data coming from AMI on the field help to create a very accurate model, improving the fidelity of the model for power system studies.
- But nothing is for free…

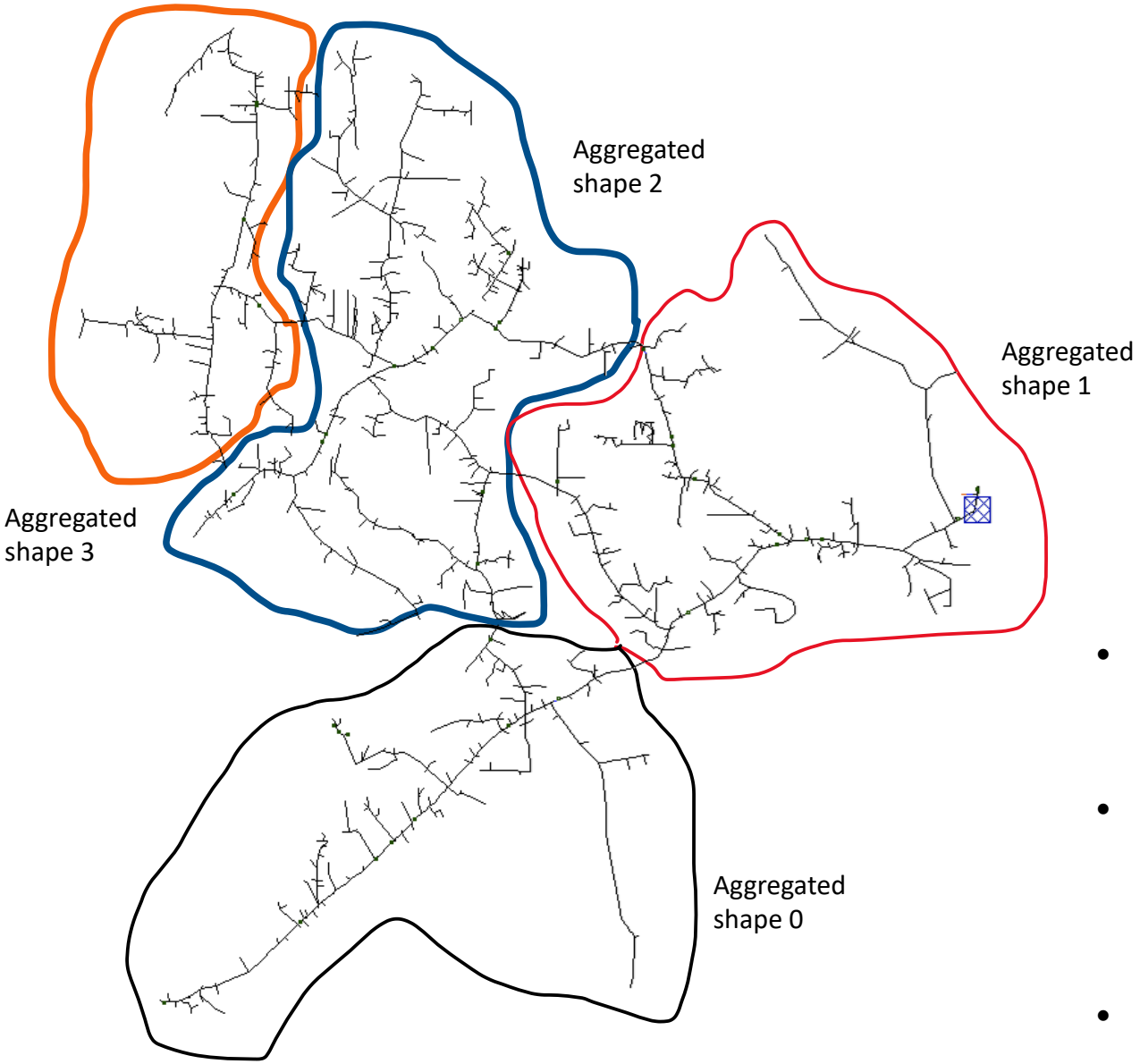
Data diversity within a distribution model

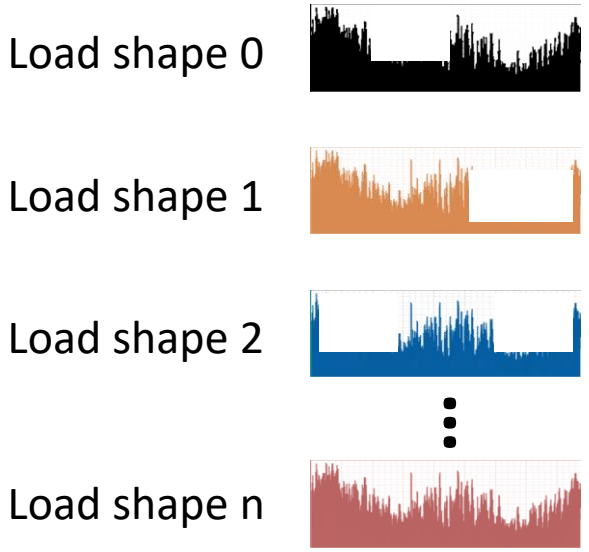




- Large amount of data means large amount of memory.
- Parallelization becomes an issue in this case, limiting the simulation capabilities when working with standard computer architectures.
- We need to transform data into information.

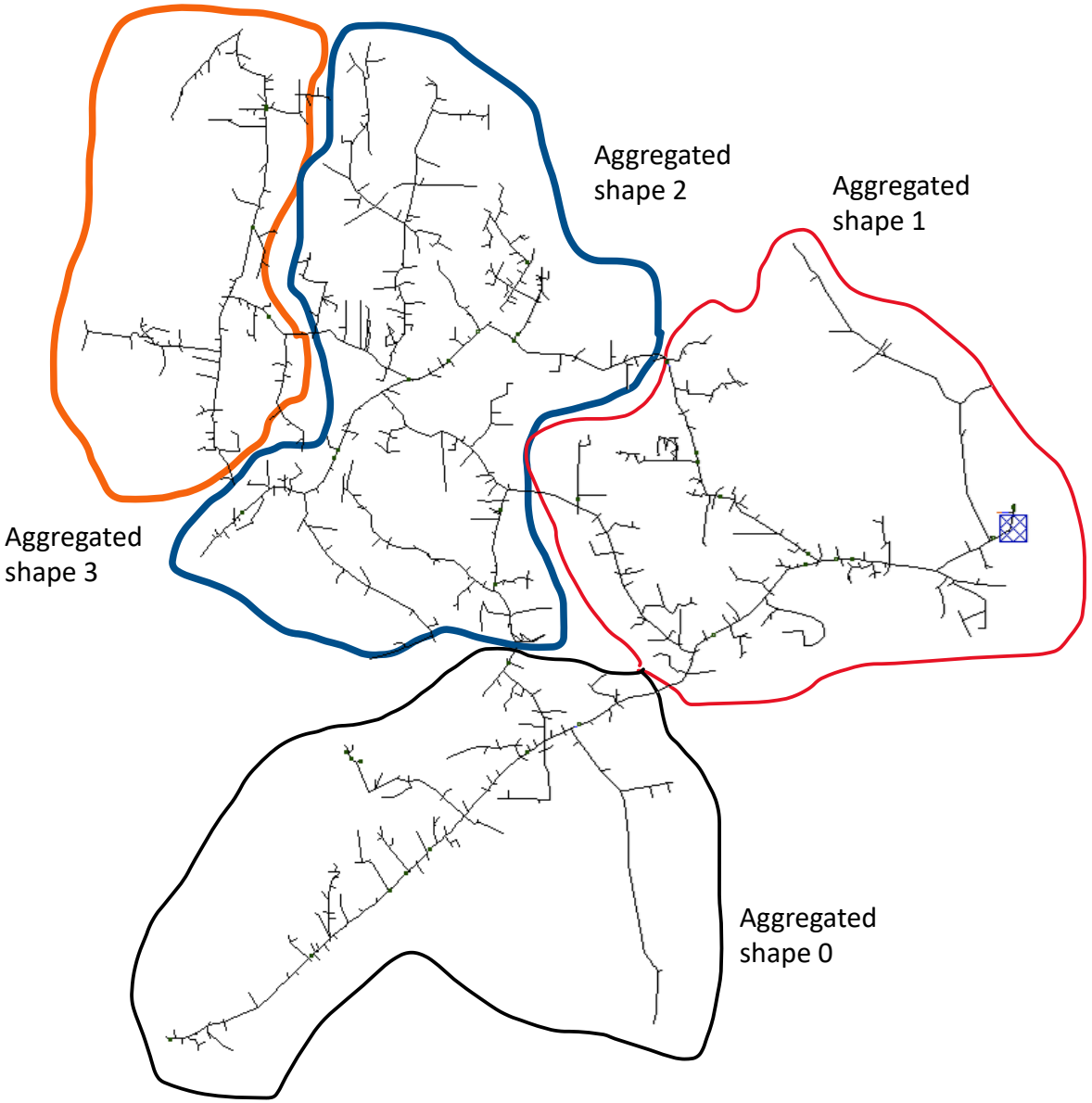
Making the model lighter





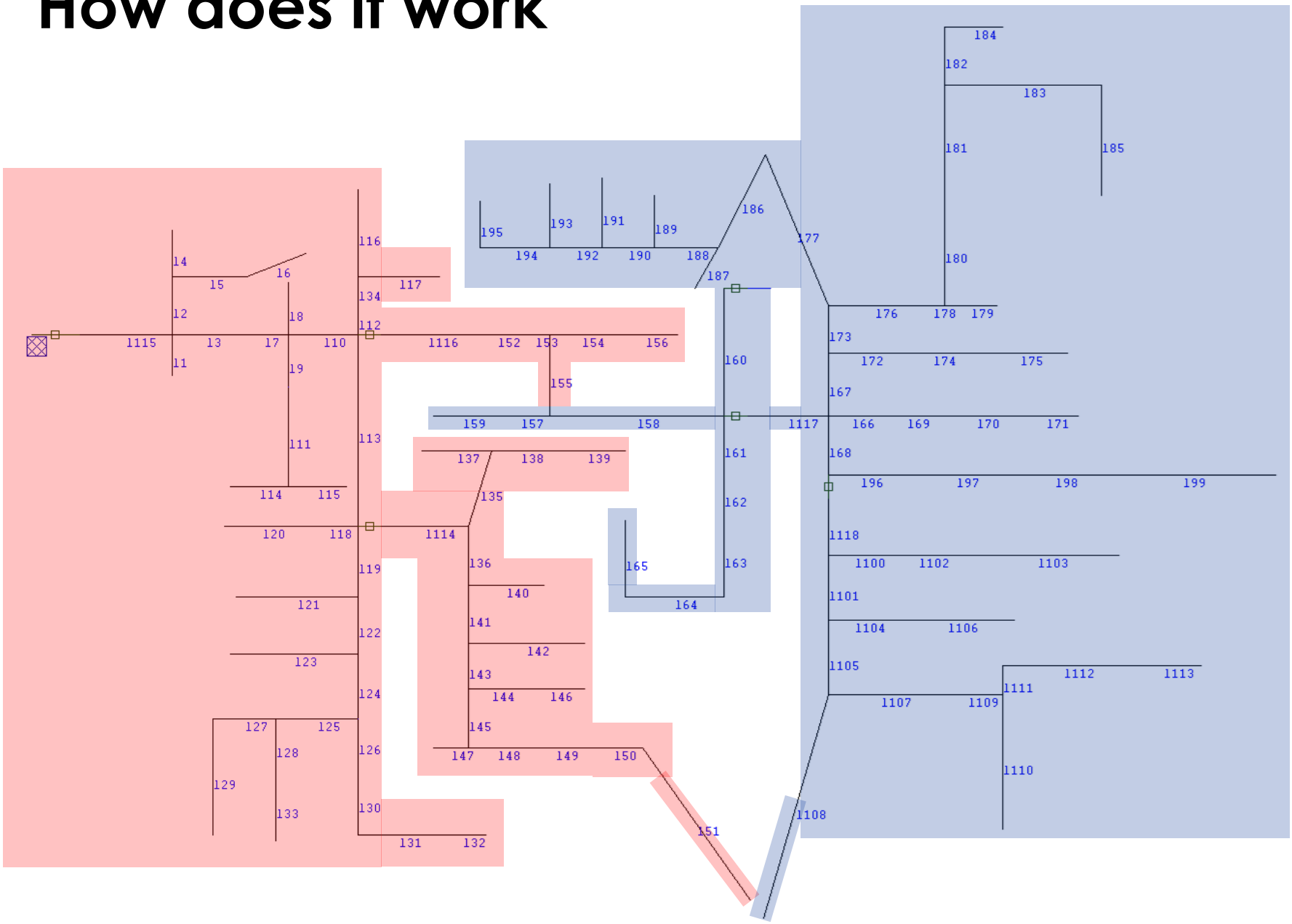
- Splitting the model into zones for aggregating load profiles sounds like a promising alternative.
- By measuring the power behavior at the head of the zone, we can estimate an approximate shape that will work for everybody within the zone.
- It must be flexible enough to reflect accuracy.

How does it work



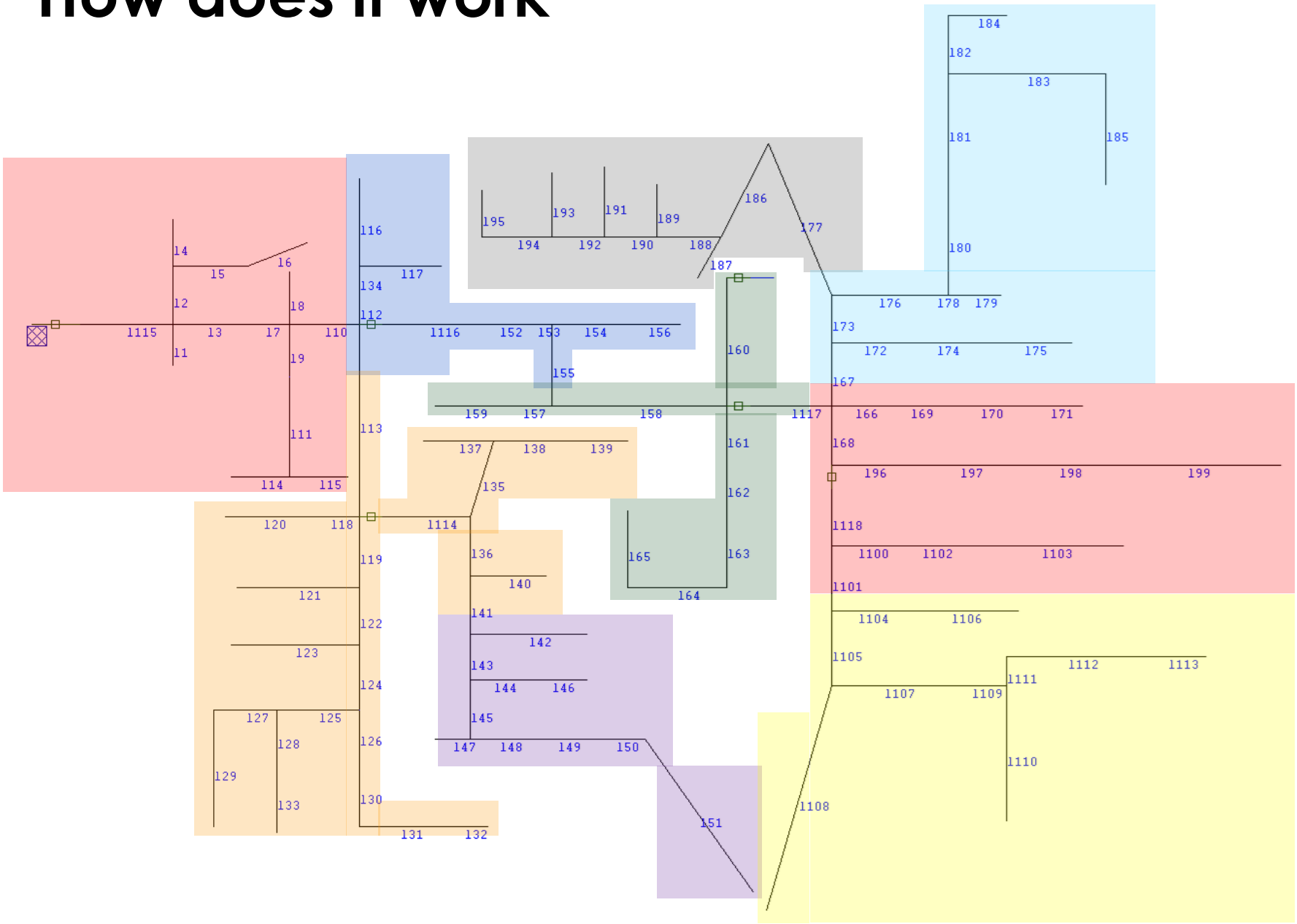
- OpenDSS uses MeTIS for tearing the model in order to obtain a symmetrical distribution based on the number of buses.
- MeTIS was integrated into OpenDSS when developing the A-Diakoptics suite.
- After creating the zones, OpenDSS will add monitors to the head of each zone and run a yearly simulation to check on how the power profile looks at each point.
- With the new measurements creates an aggregate load profile that is then linked to the loads within each zone.
- The accuracy of this approach depends on the number of zones created; this number of zones is customizable for the user. The user decides if the aggregation is good enough or if it requires more zones to be considered.

How does it work



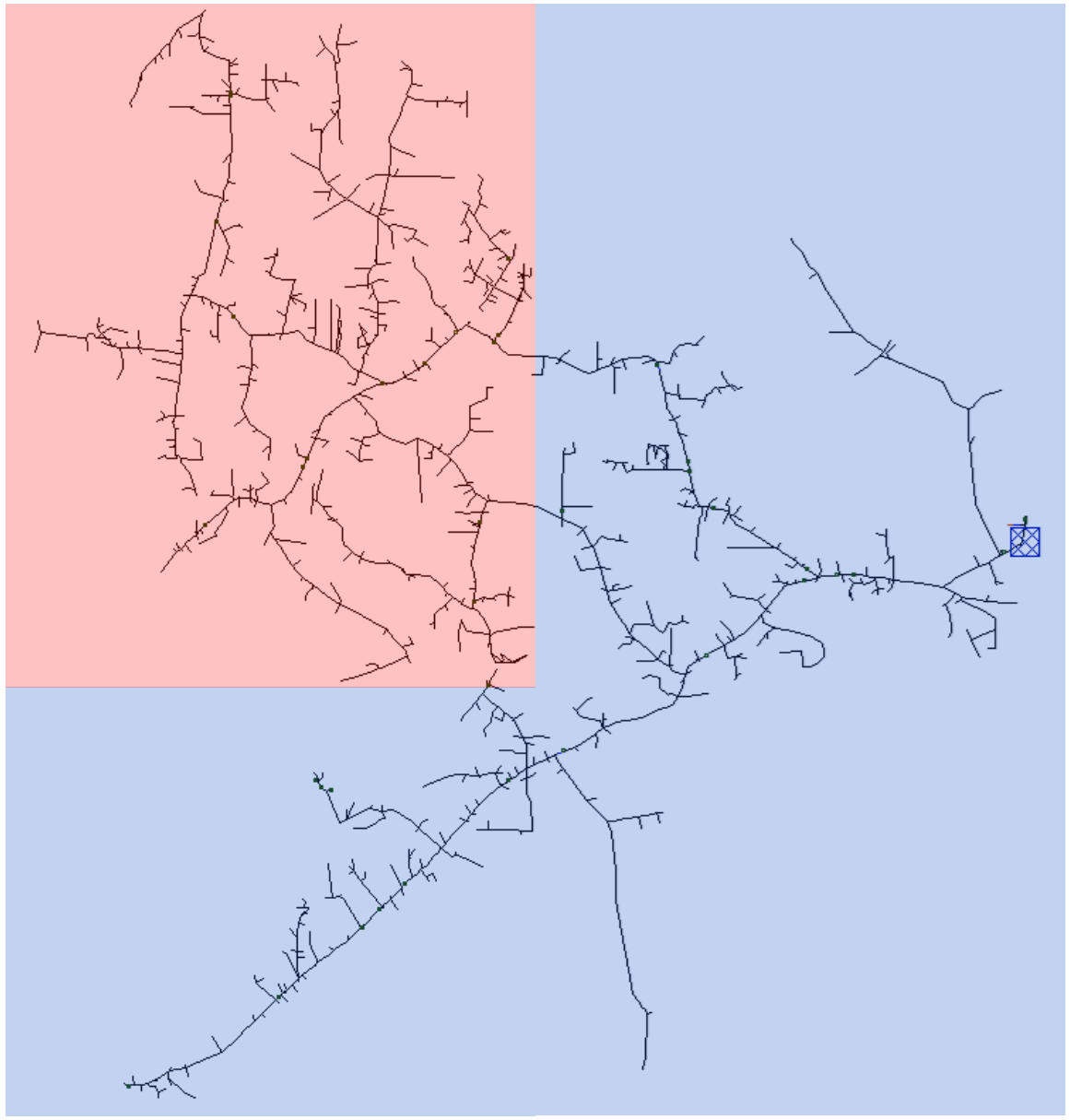
Example tearing the IEEE123 test model, 2 zones

How does it work



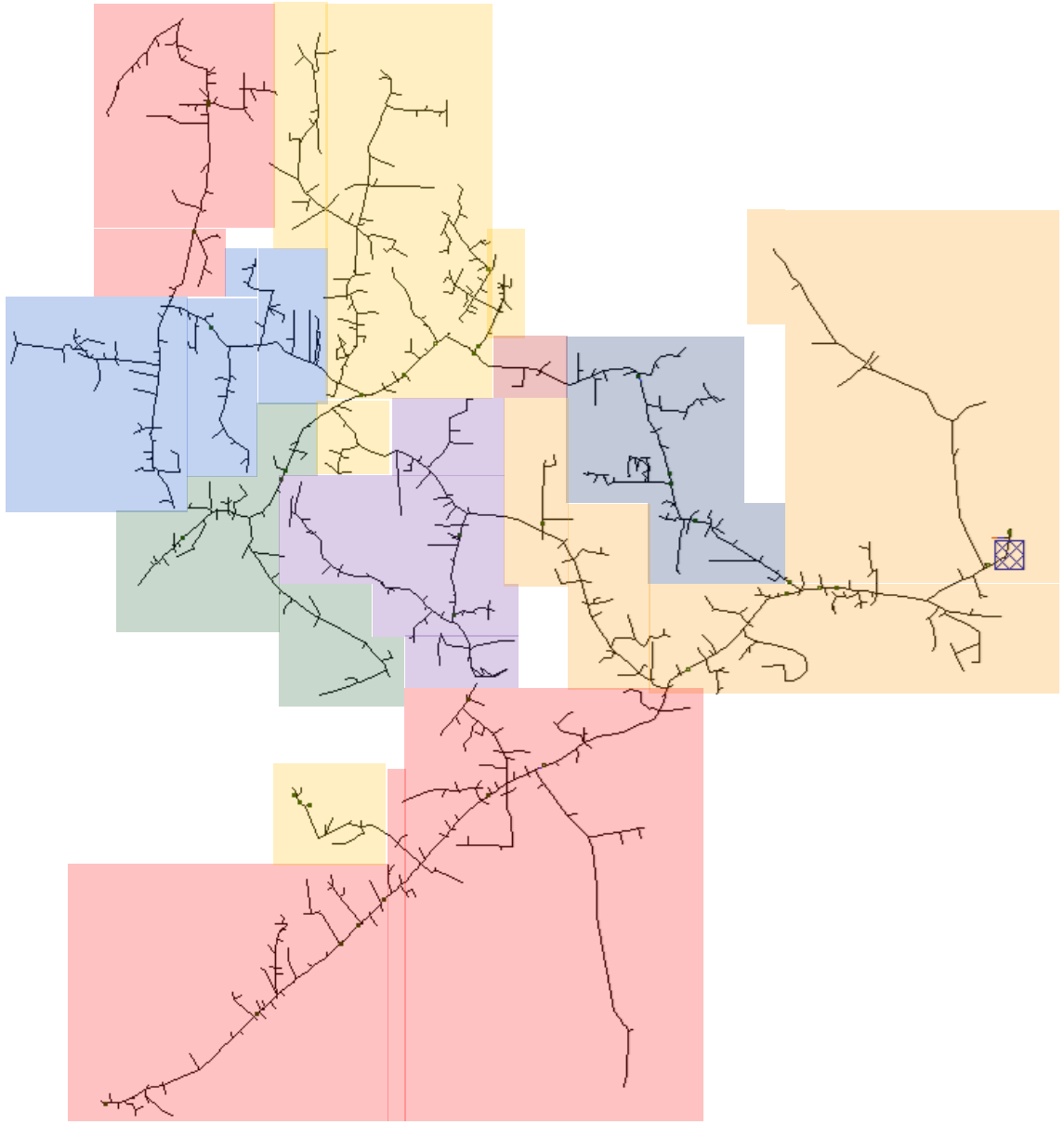
Example tearing the IEEE123 test model, 9 zones

How does it work



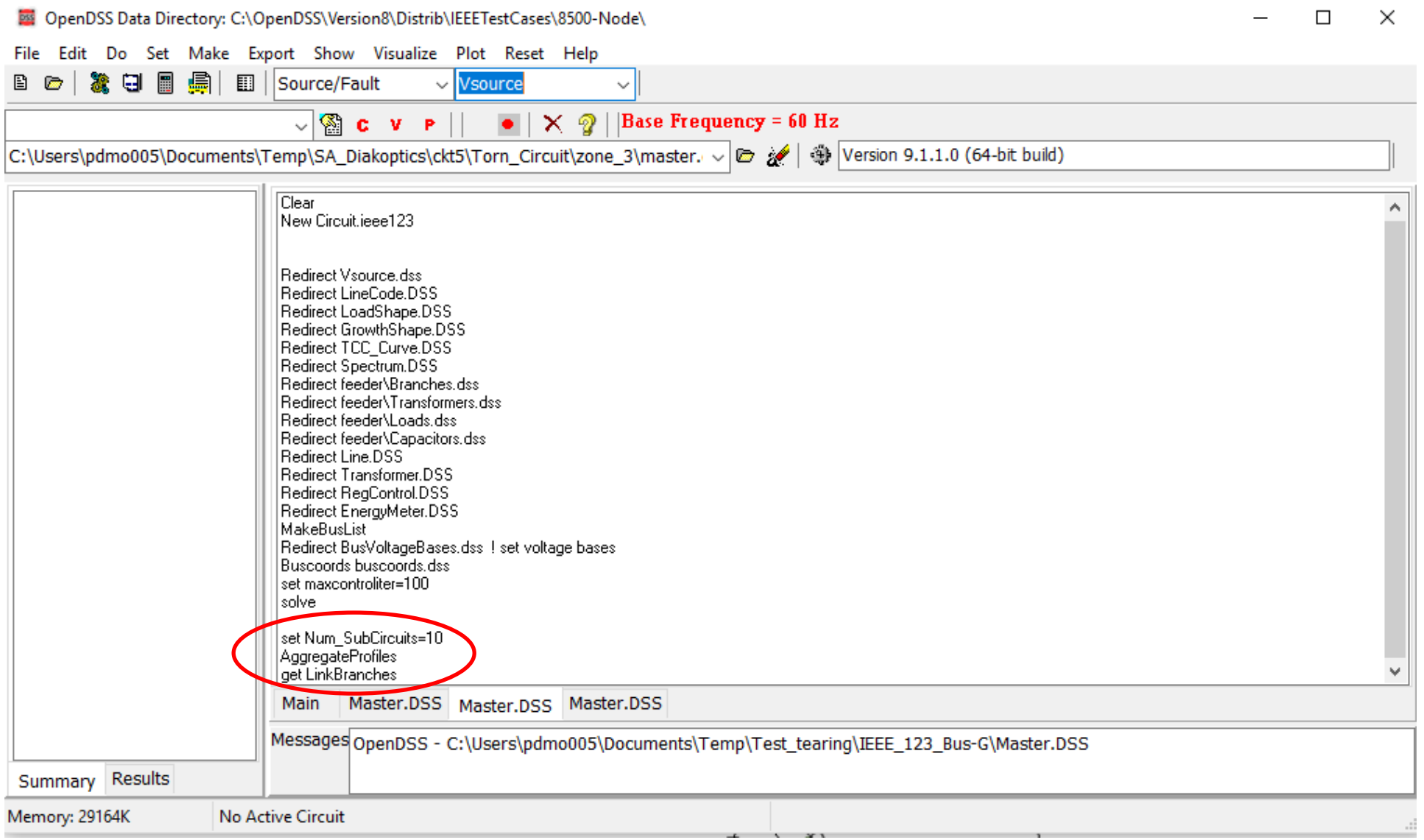
Example tearing the IEEE 8500 test model, 2 zones

How does it work



Example tearing the IEEE 8500 test model, 10 zones

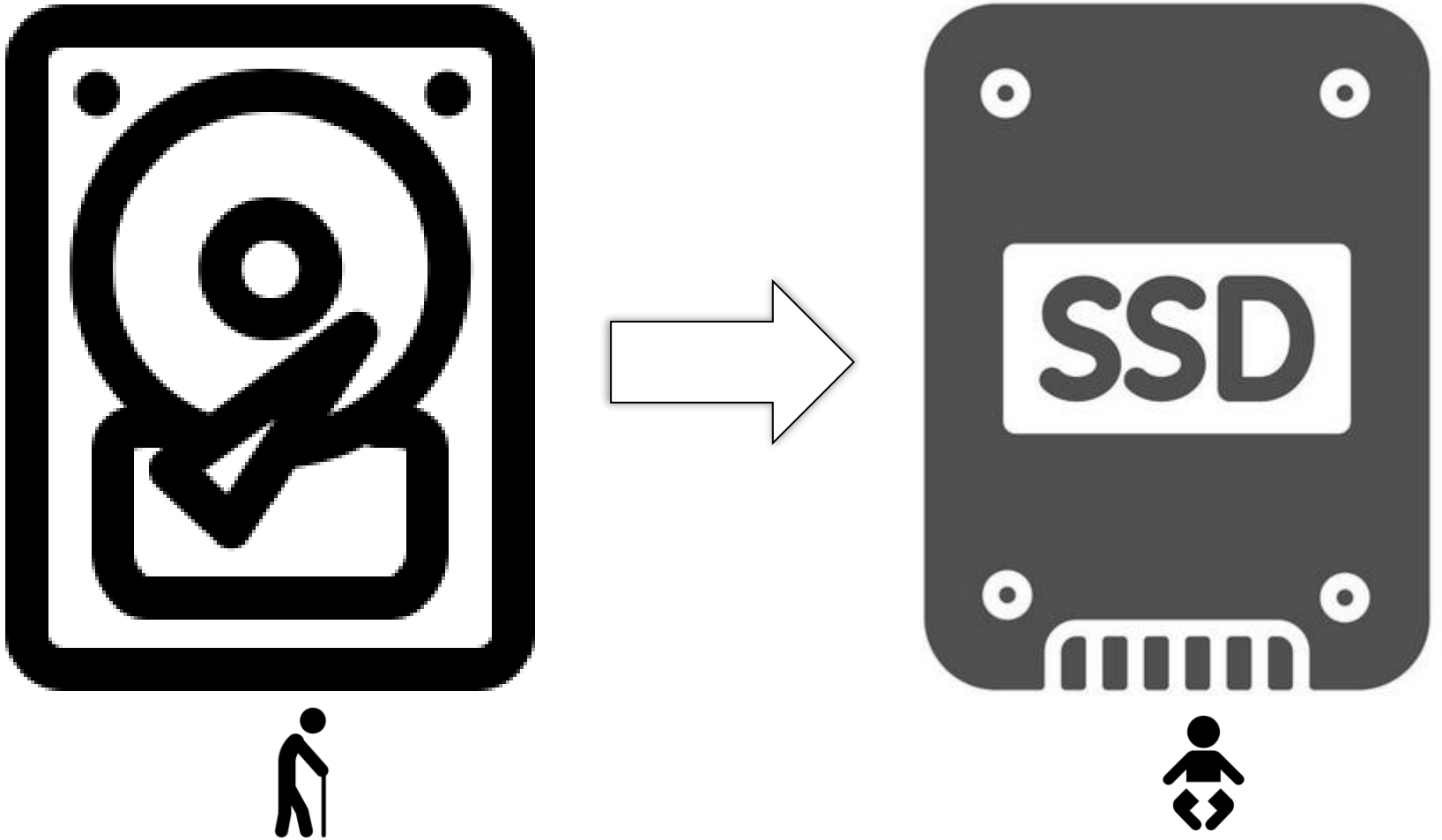
The command in OpenDSS



Memory mapping

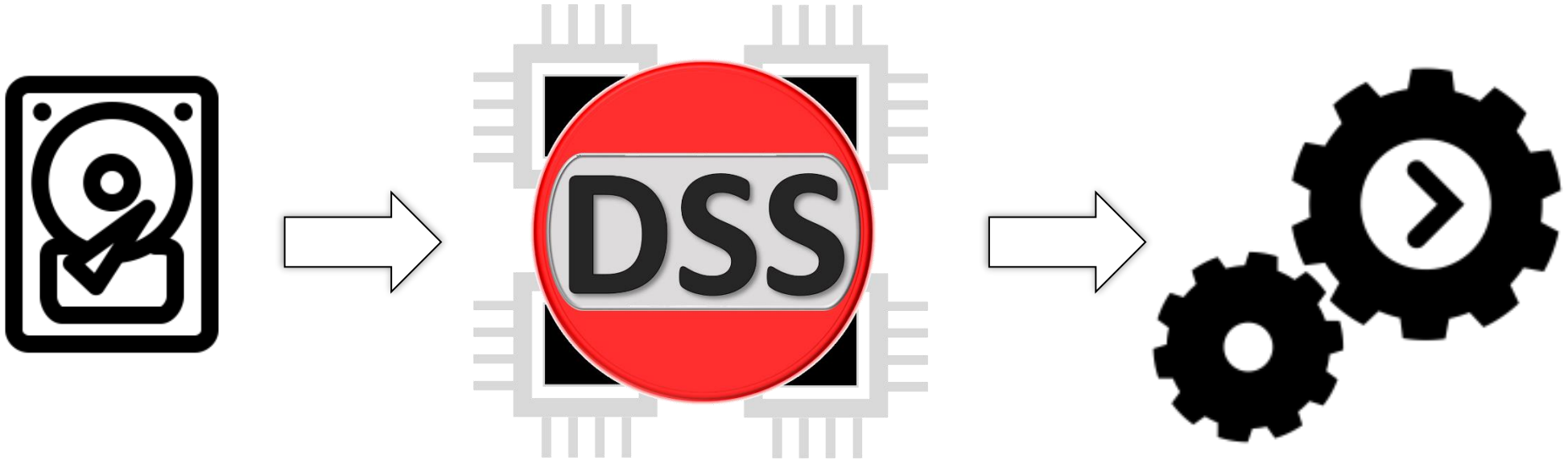
Introducing memory mapped files into OpenDSS

Catching up with technology



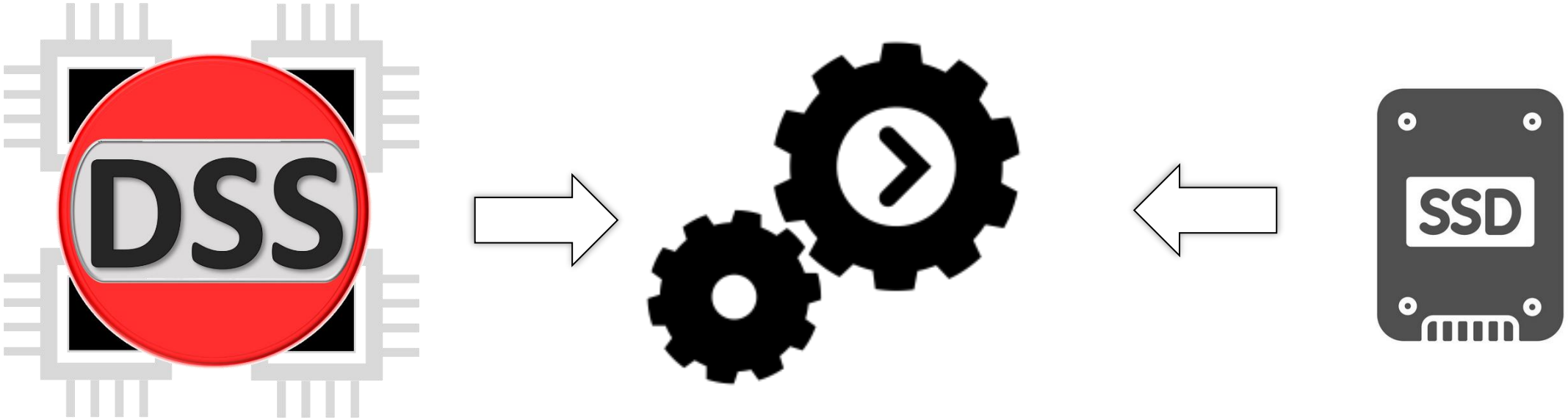
Introducing memory mapped files into OpenDSS

Catching up with technology



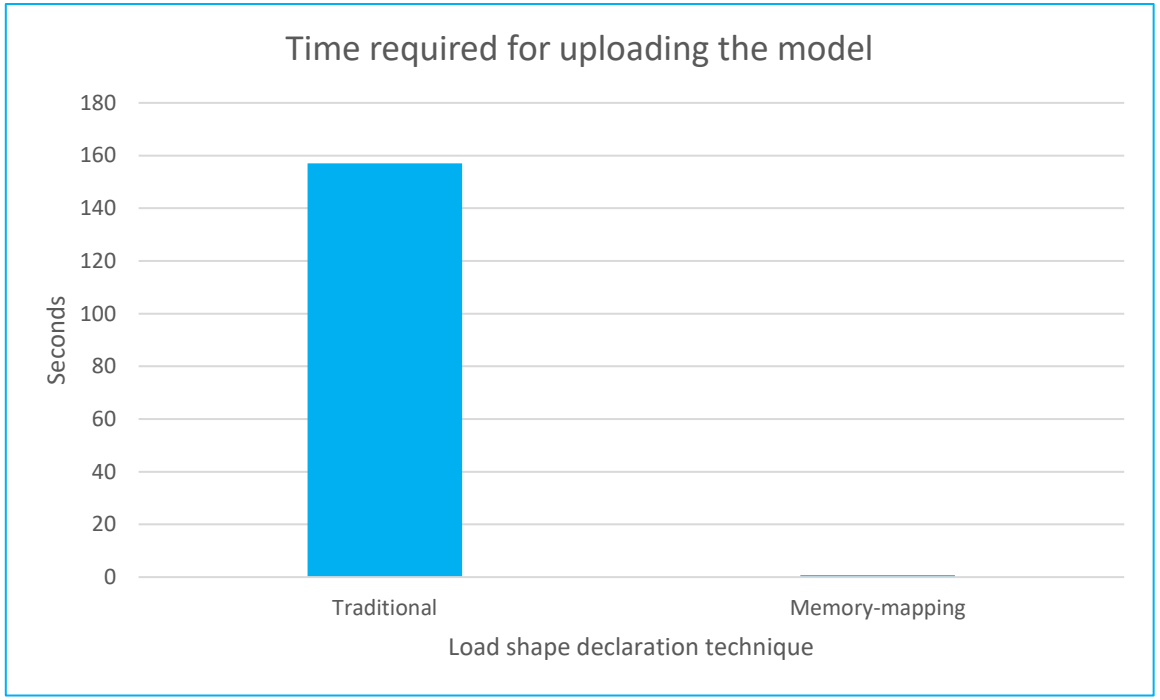
Introducing memory mapped files into OpenDSS

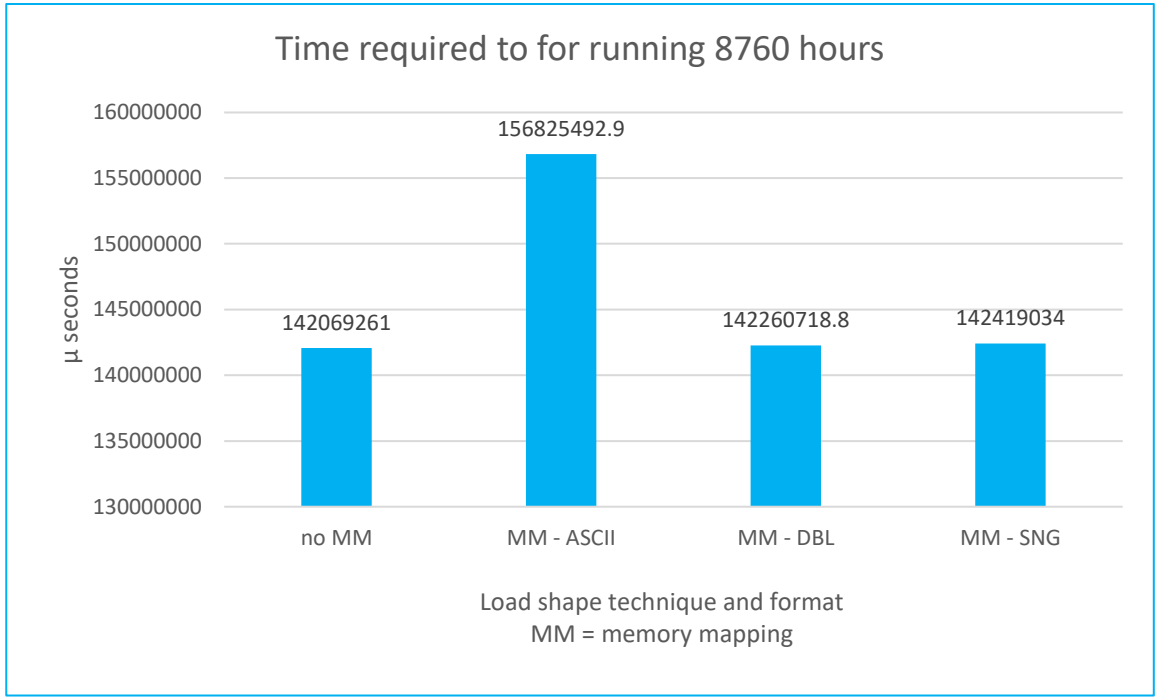
Catching up with technology



Introducing memory mapped files into OpenDSS

Catching up with technology





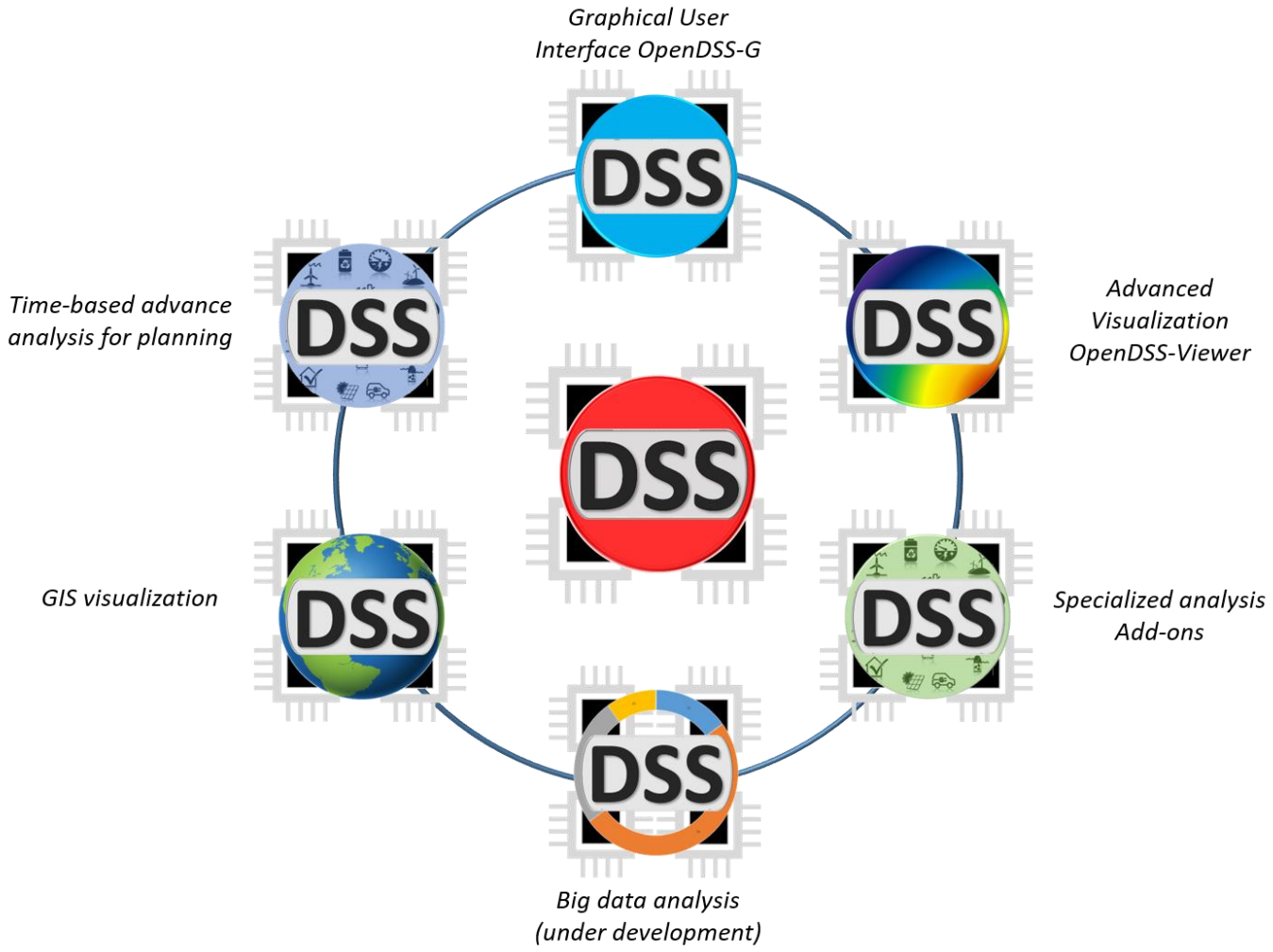
New Loadshape.LS_PhaseA npts=8760 interval=1 MemoryMapping=Yes mult=(file=myFile.txt)

OpenDSS-G

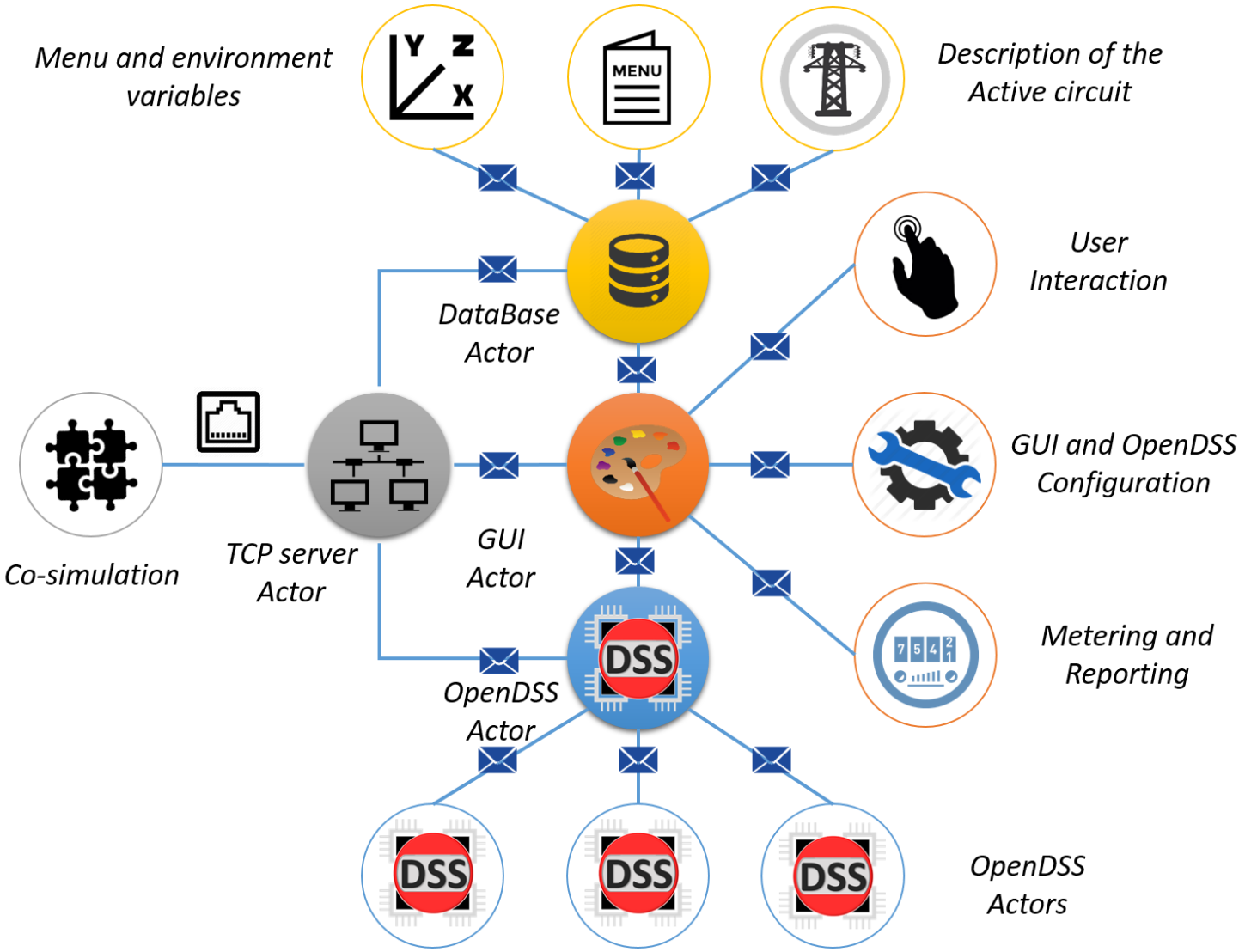
OpenDSS-G YouTube channel

https://www.youtube.com/channel/UCGe58SDH3Iq-EGvnxEOuWaQ

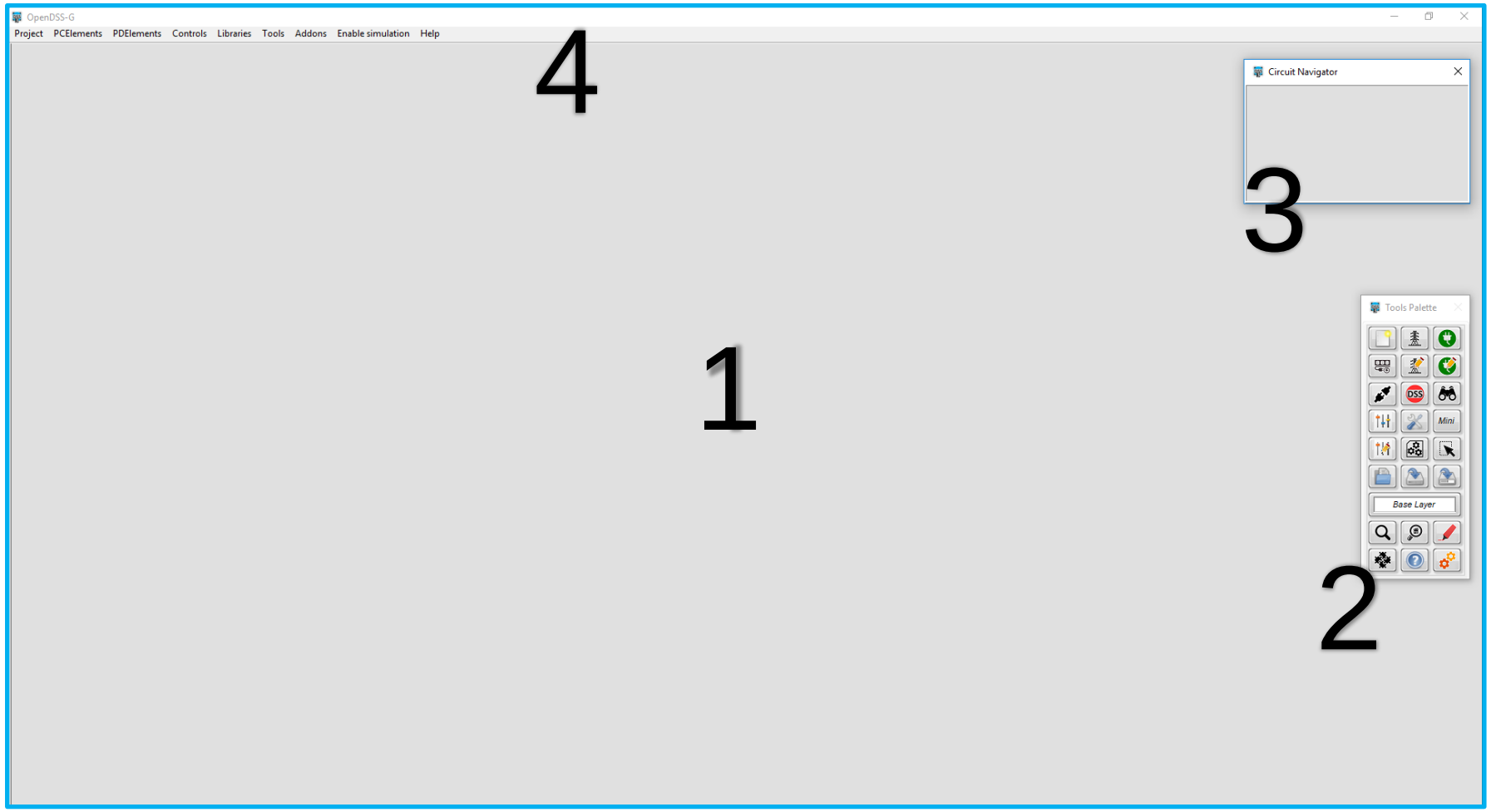
OpenDSS derivative products



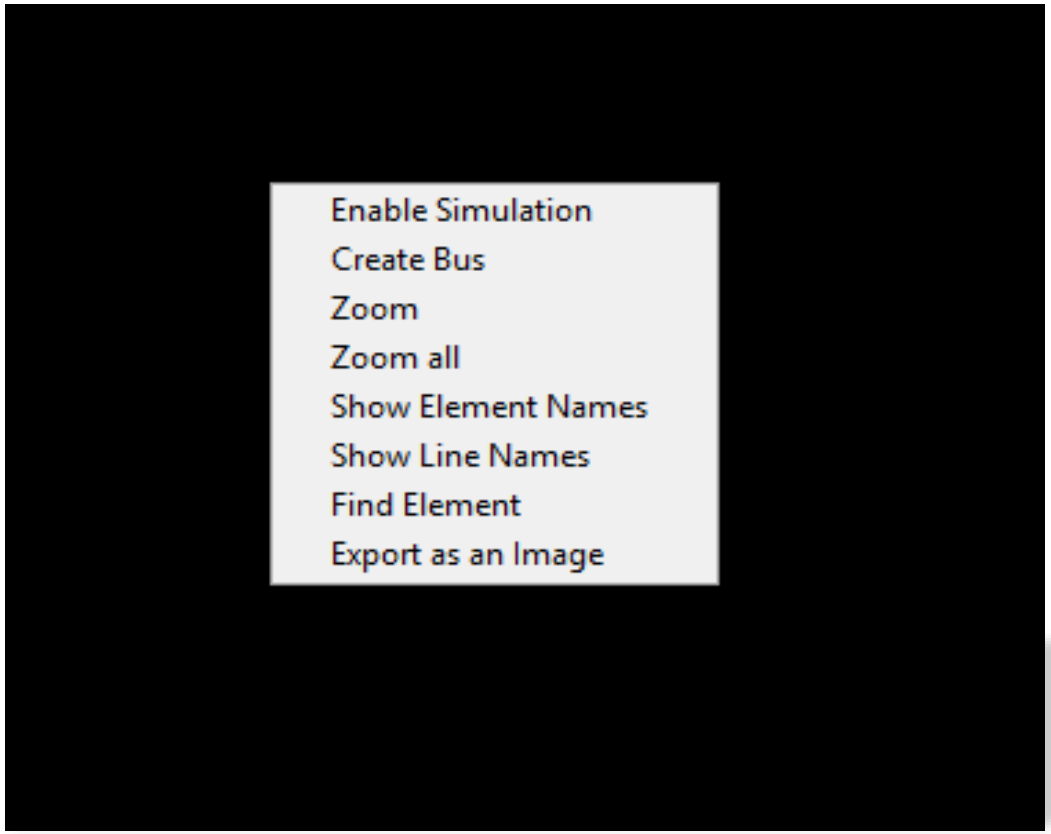
Introduction to OpenDSS-G

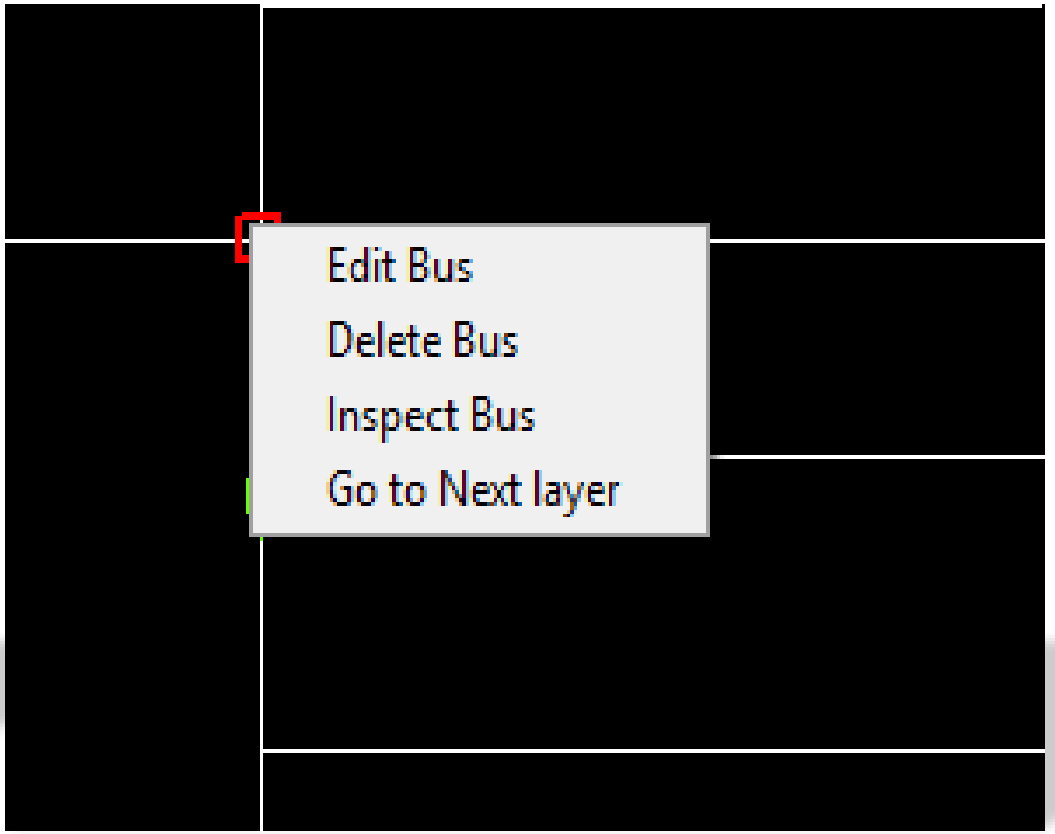


Introduction to OpenDSS-G

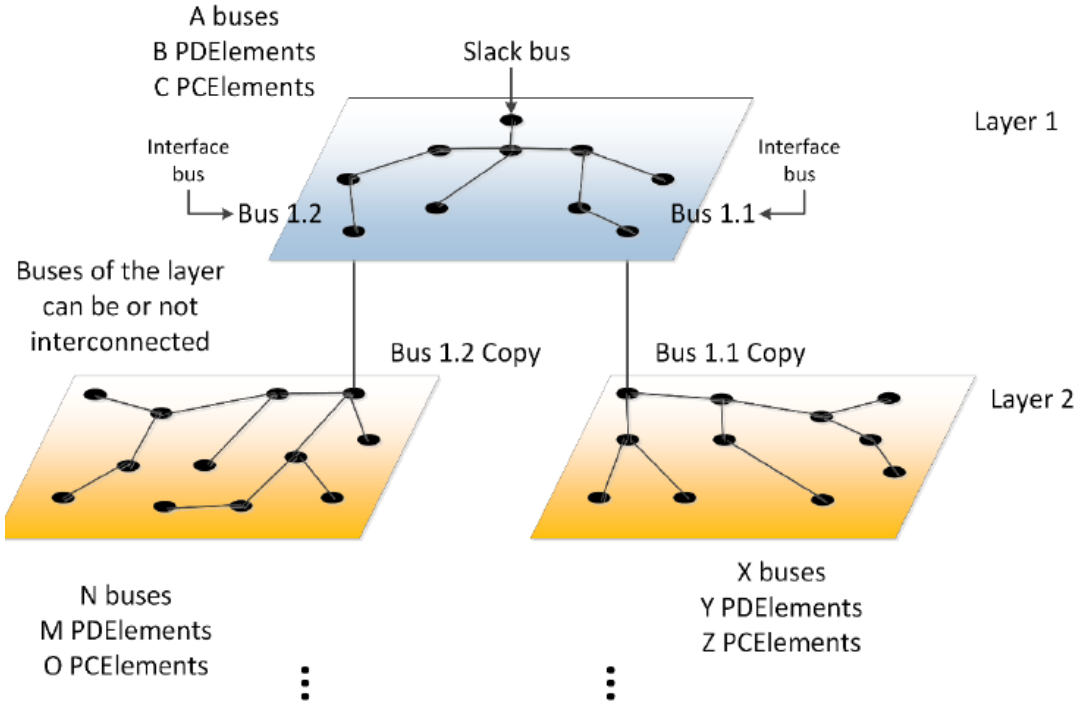


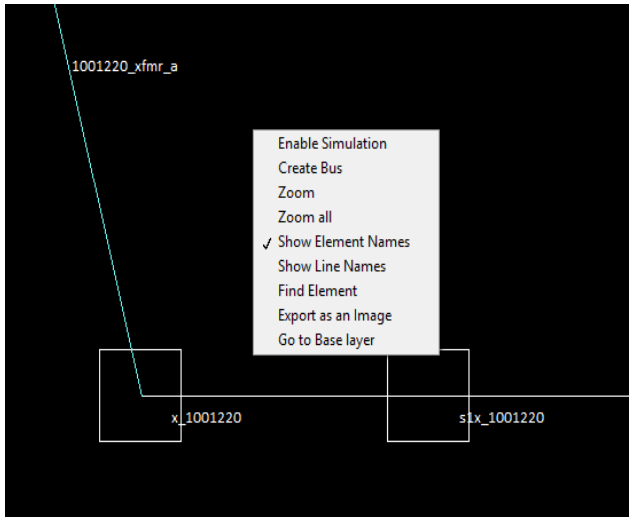
Introduction to OpenDSS-G

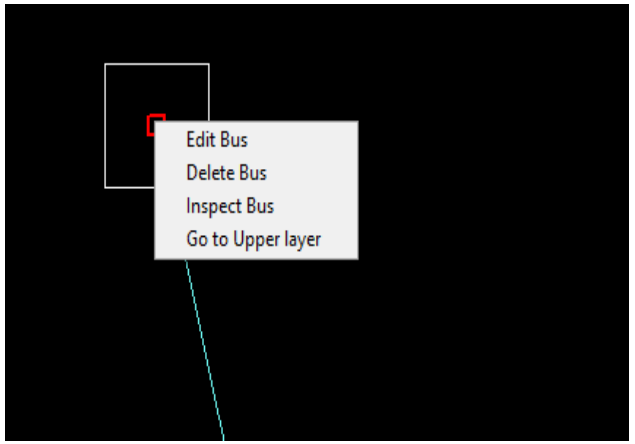




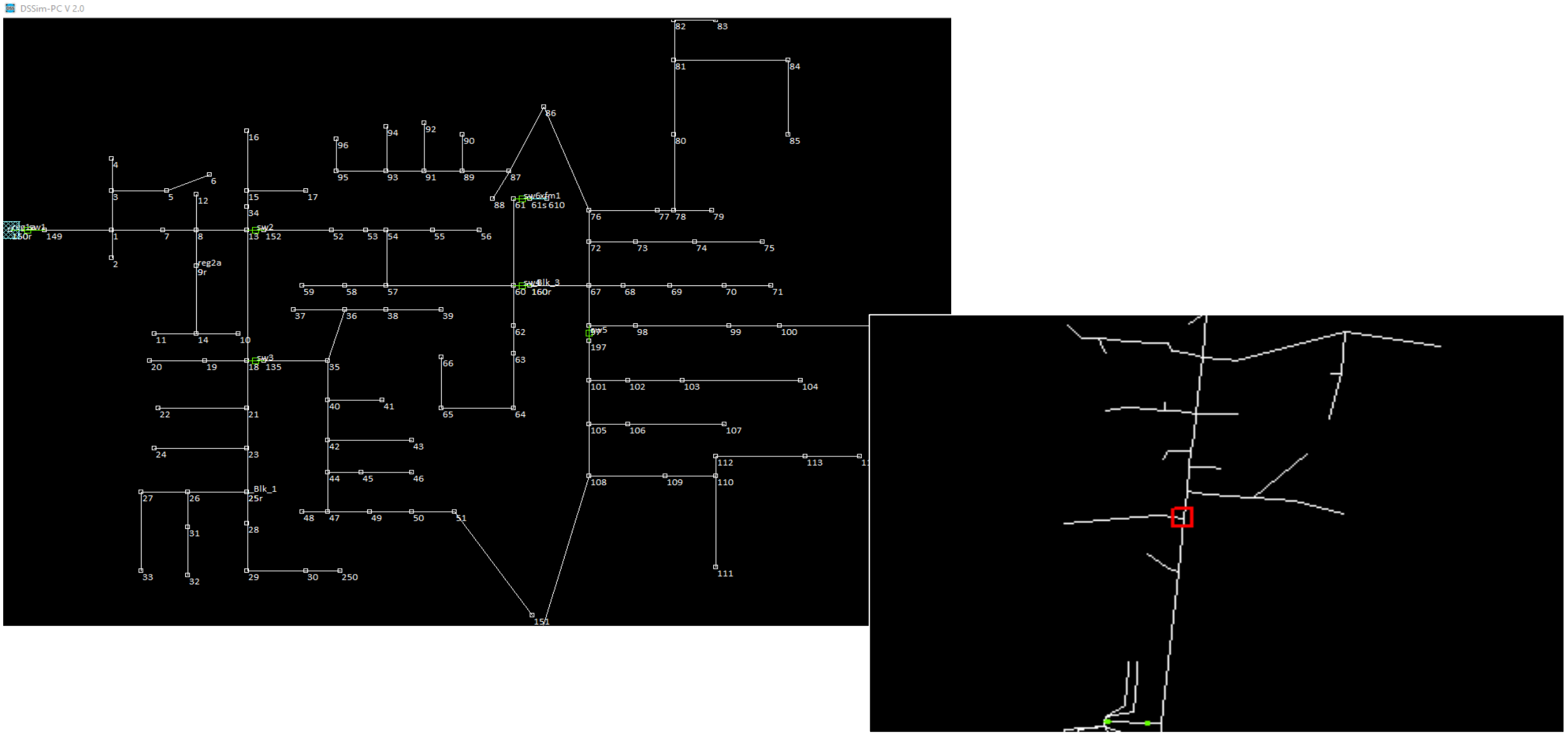
Introduction to OpenDSS-G





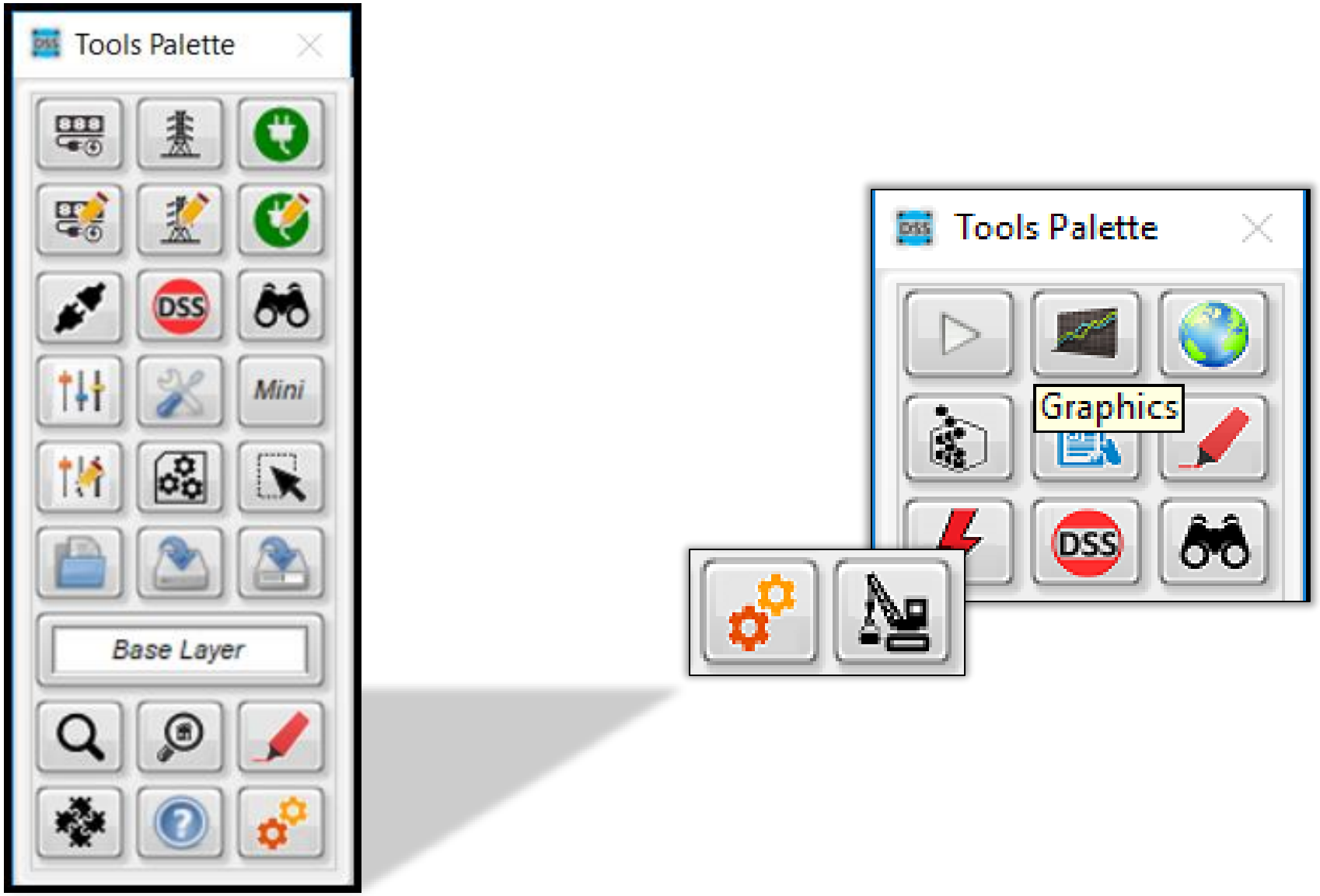


Introduction to OpenDSS-G

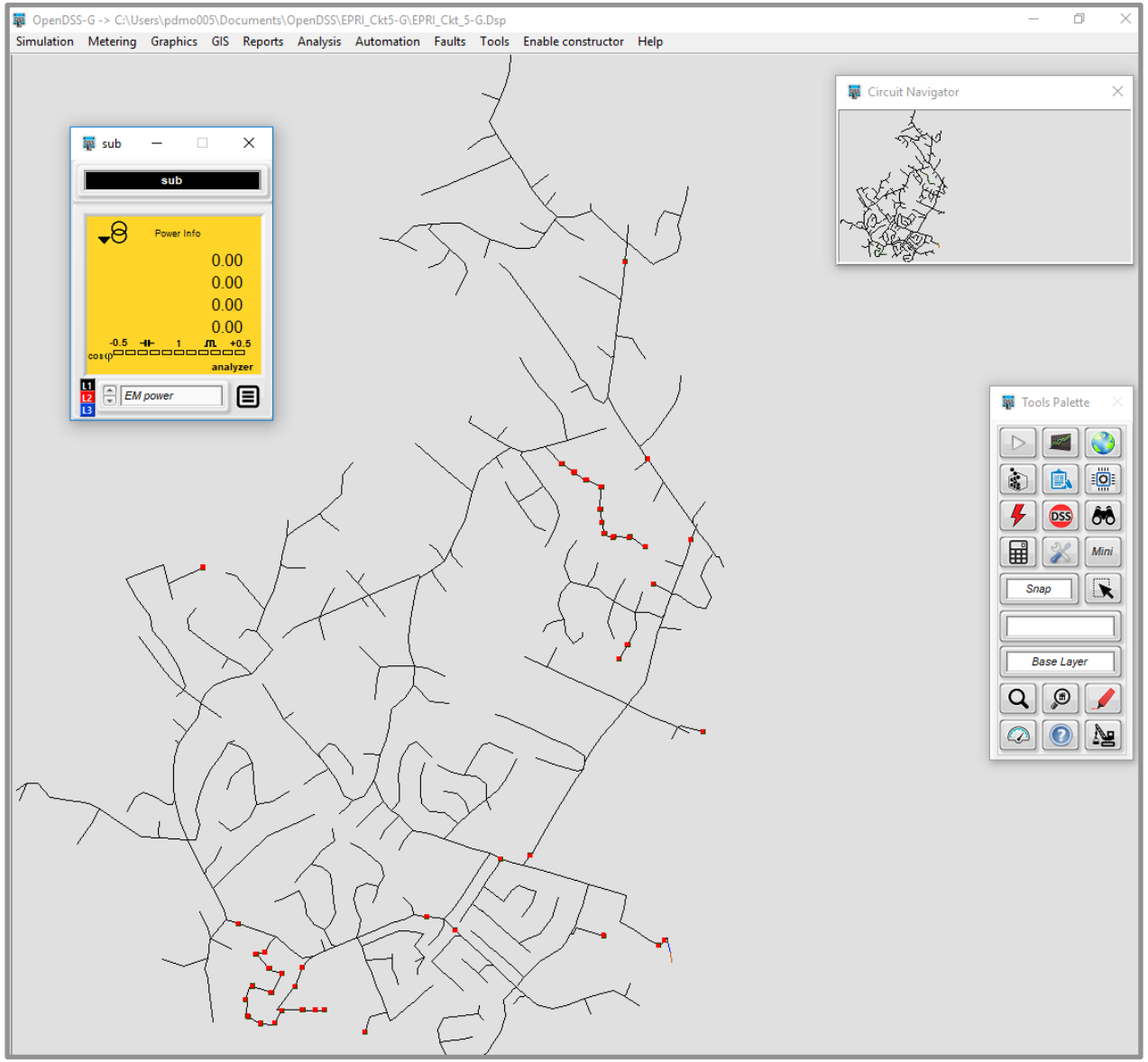


Introduction to OpenDSS-G

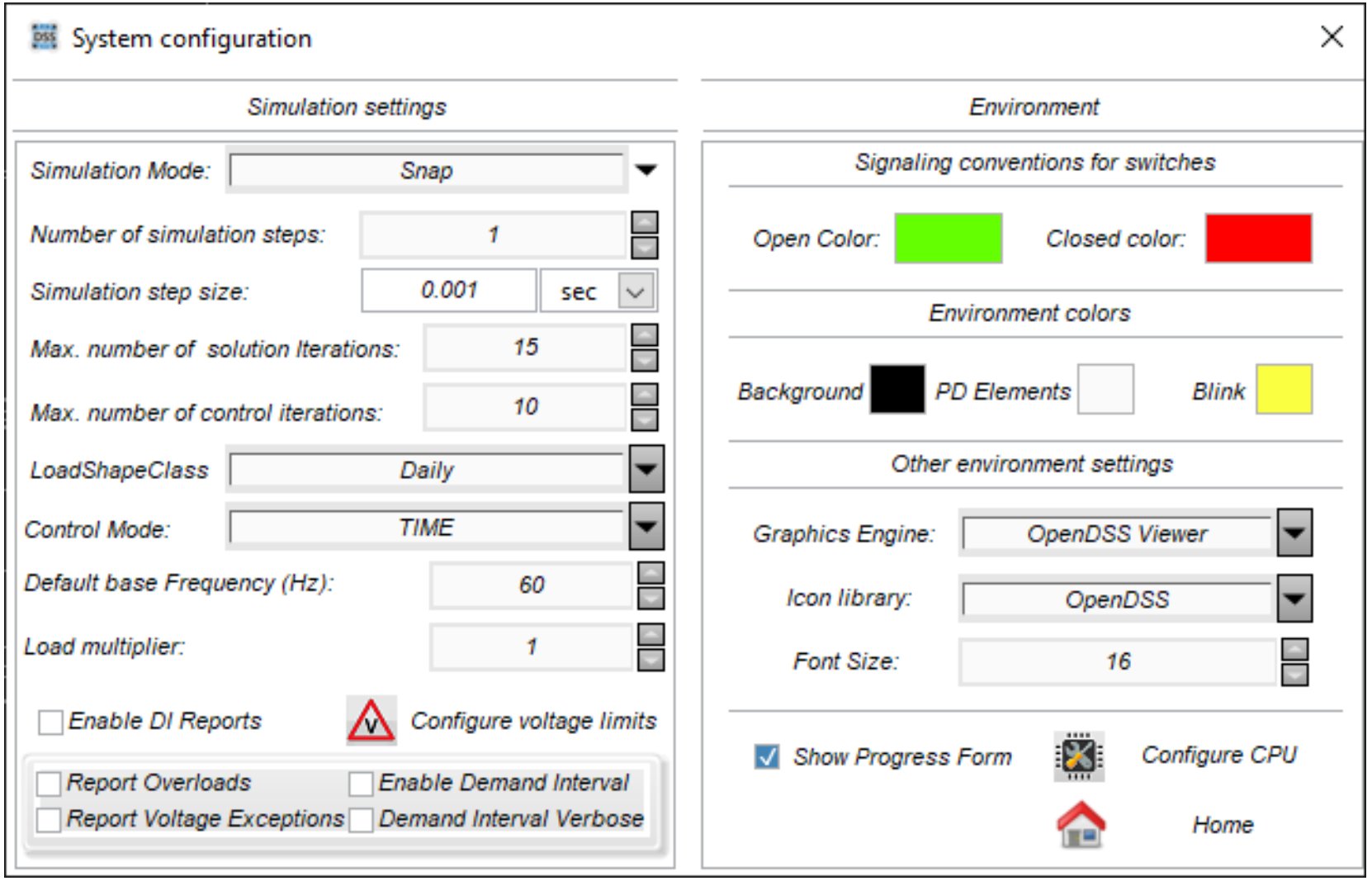
Introduction to OpenDSS-G



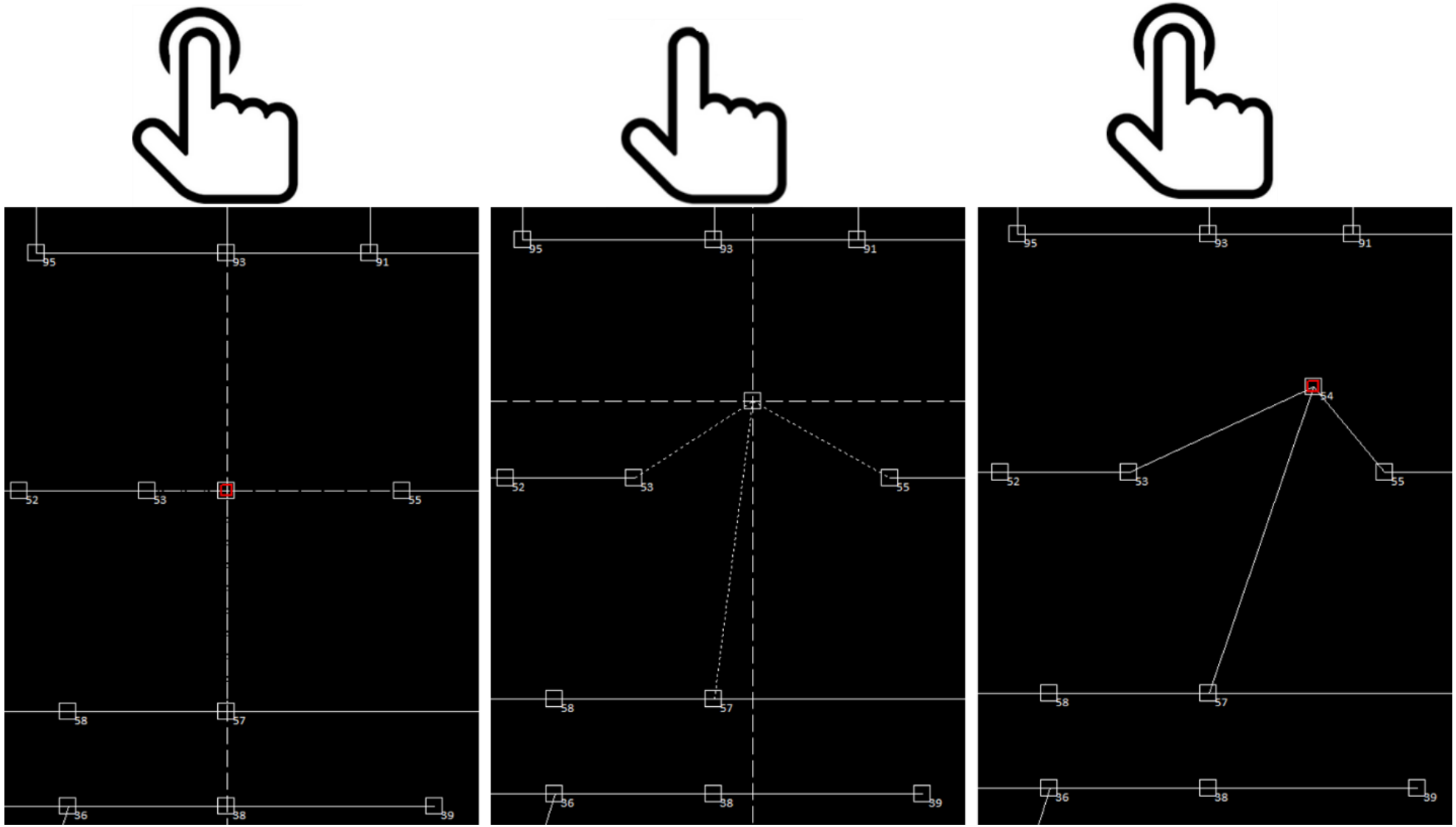
Introduction to OpenDSS-G



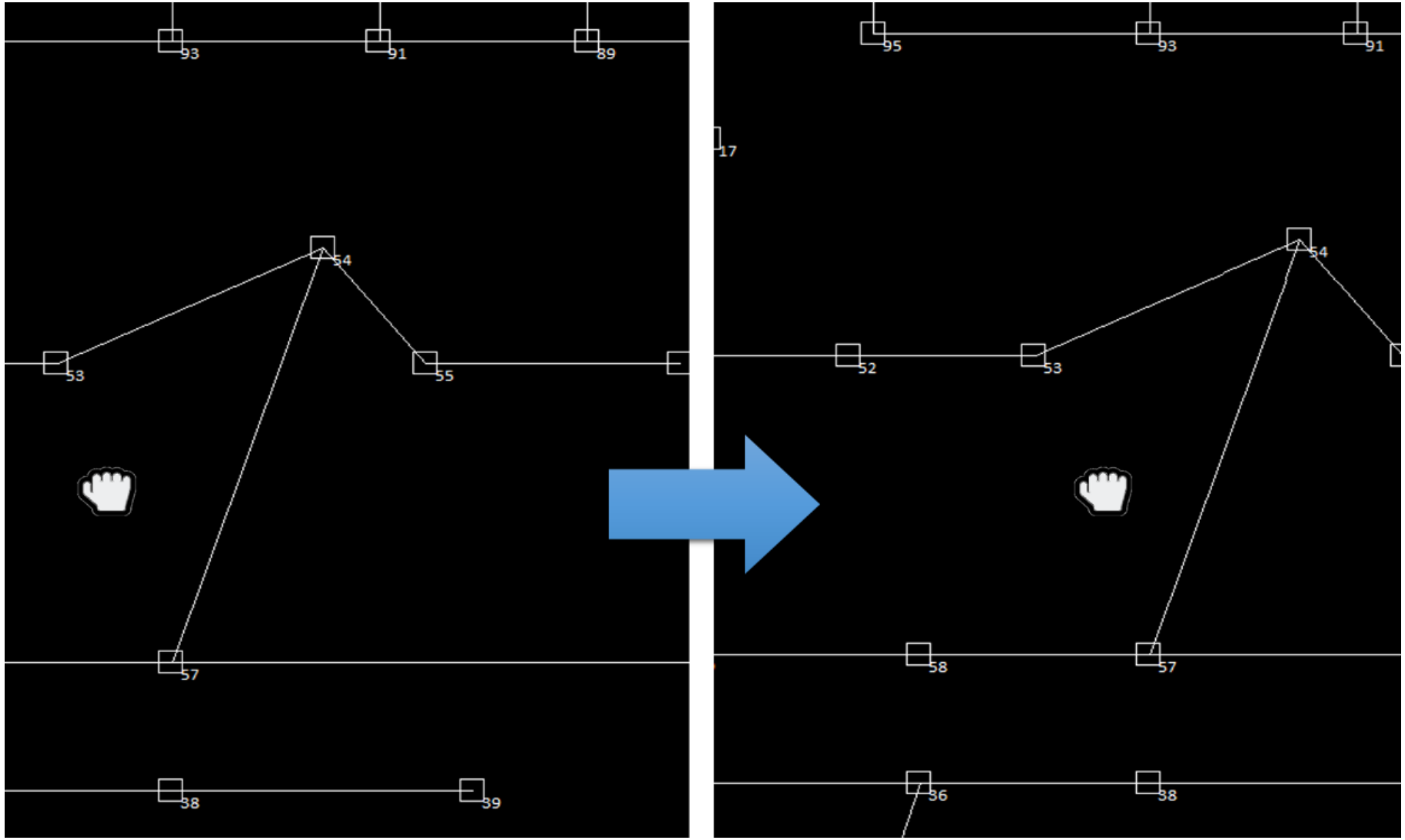
Introduction to OpenDSS-G



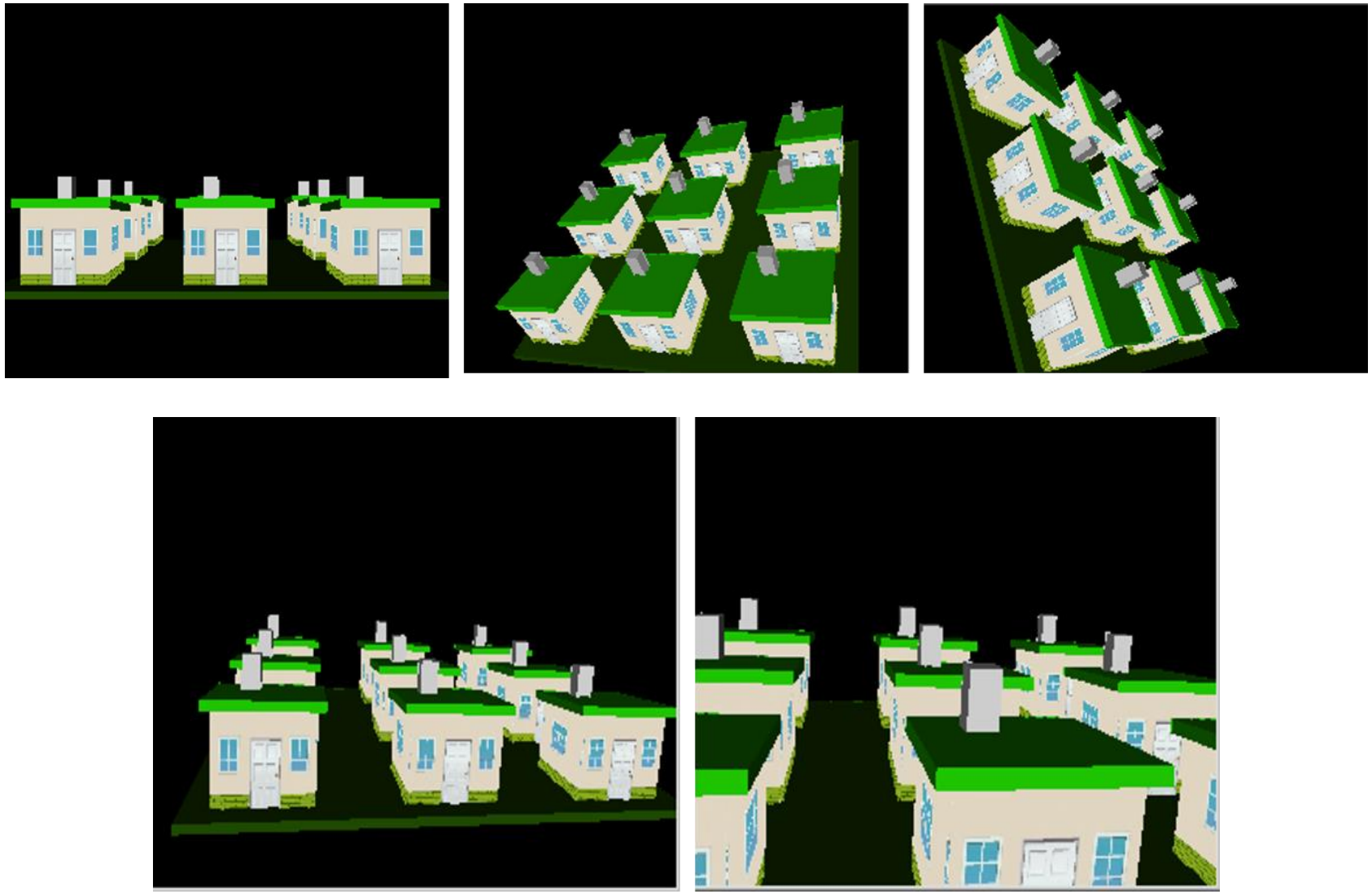
Introduction to OpenDSS-G



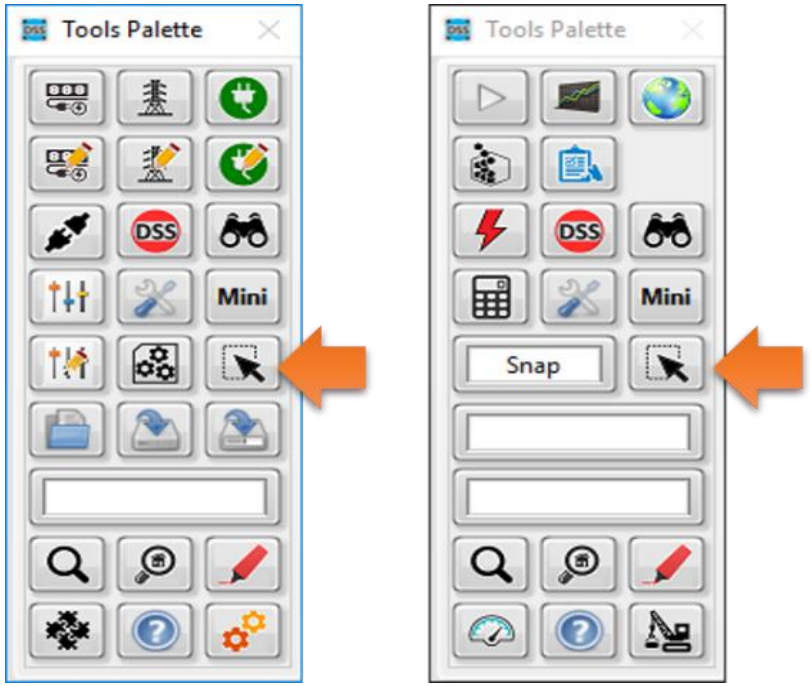
Introduction to OpenDSS-G

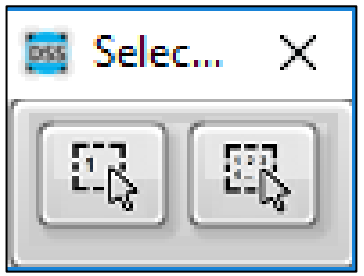


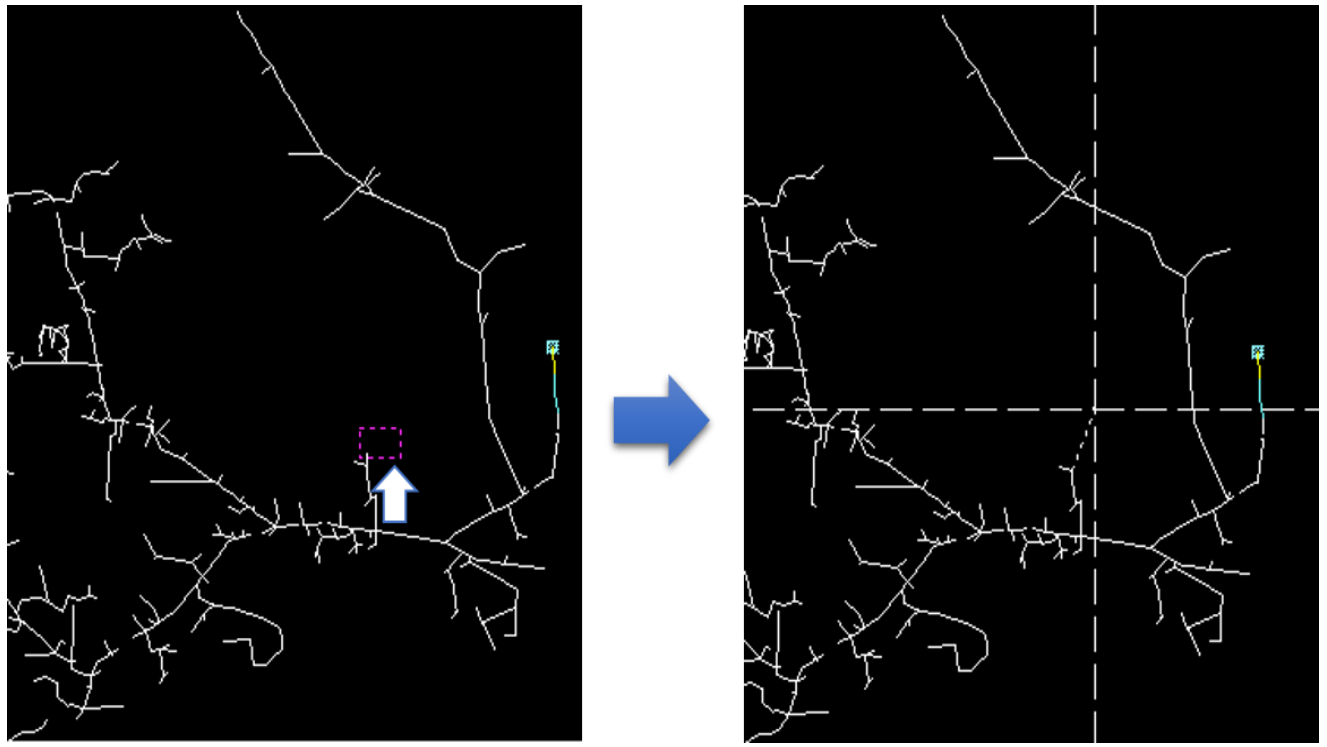
Introduction to OpenDSS-G



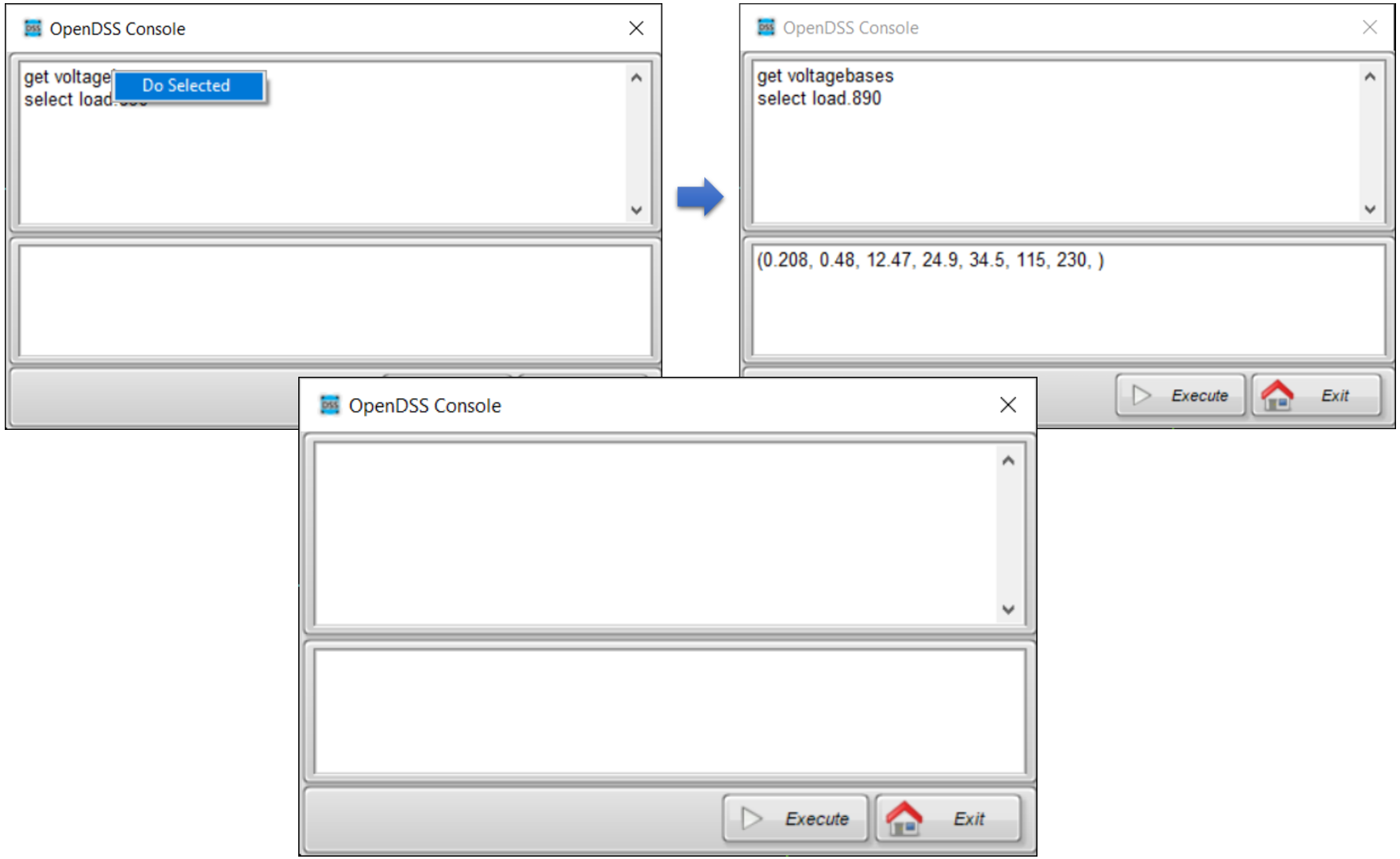
Introduction to OpenDSS-G







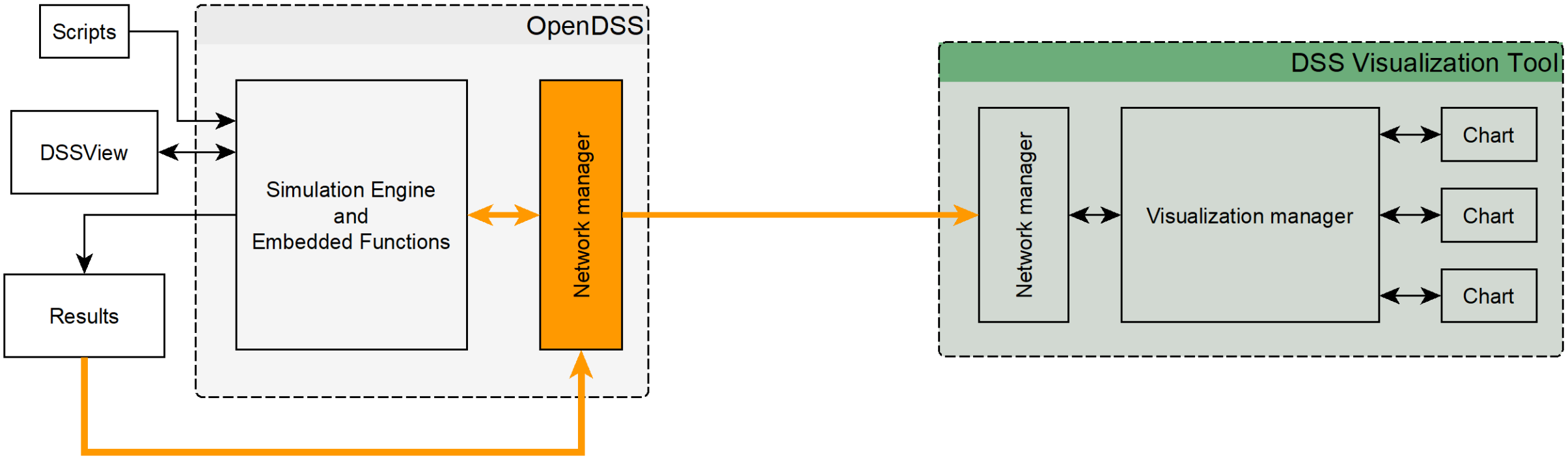
Introduction to OpenDSS-G



Complementary Tools

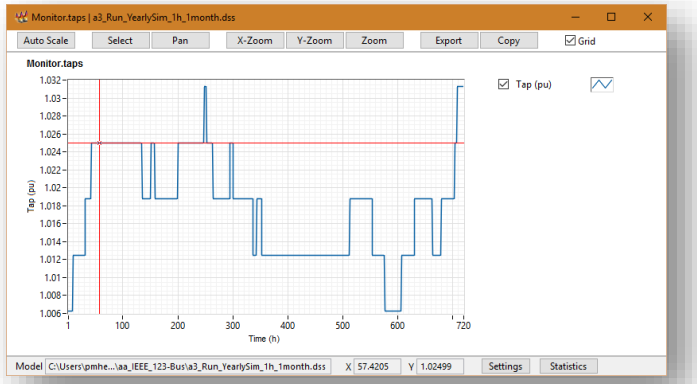
Advanced Graphics Module for OpenDSS (OpenDSS-Viewer)

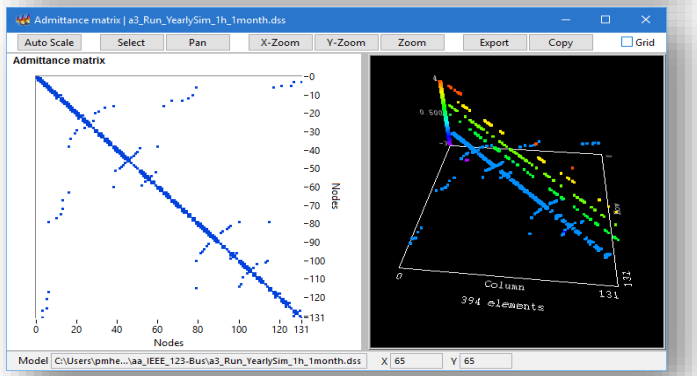
Developed by Miguel Hernandez (EPRI). Enhance the visualization of Distribution System Simulations with a **flexible**, **scalable** and **meaningful** approach.

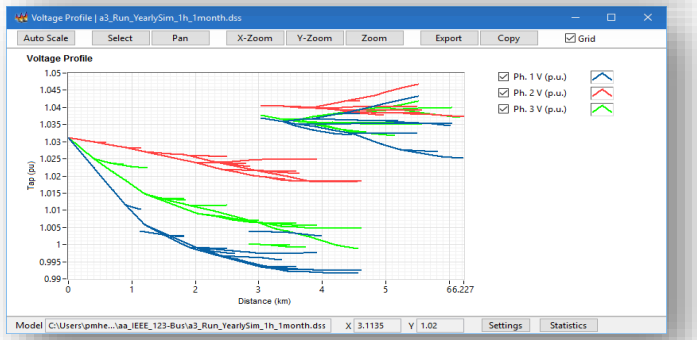


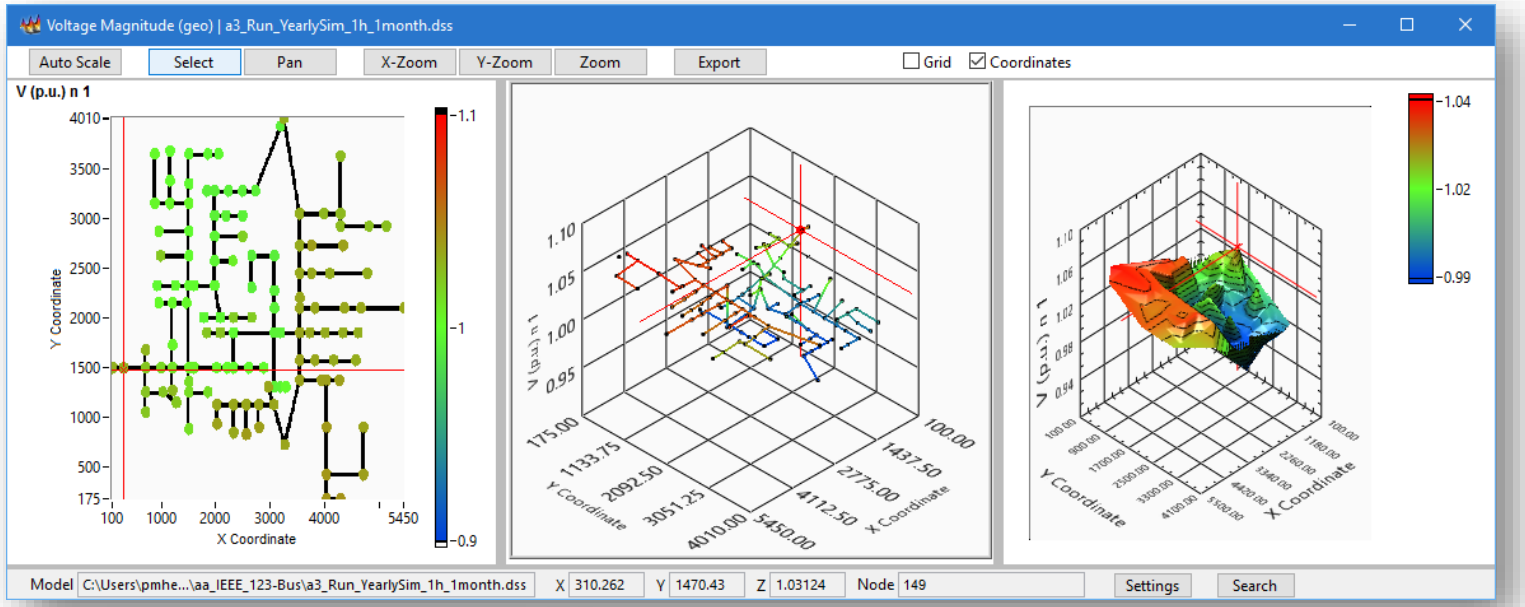
https://epri.box.com/s/o4itfn23txv73iyo3y1ukbs9nmgsspgo

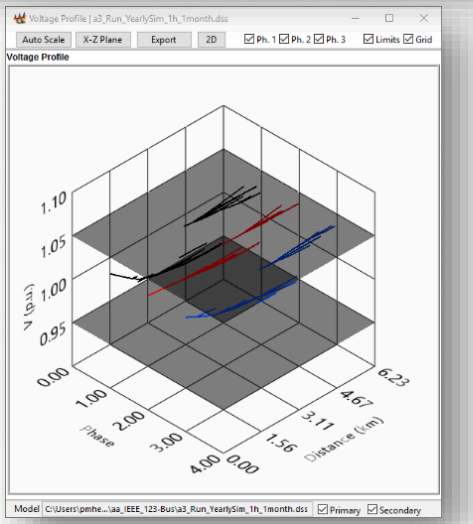
Complementary Tools

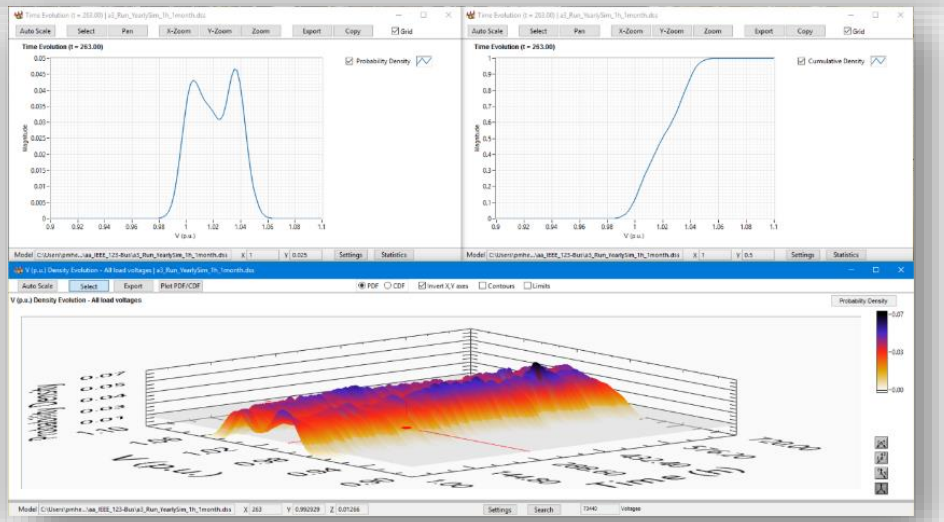












Lets see what's new in version 4

Thanks