100 Elwood Davis Road ♦ North Syracuse, NY 13212 ♦ USA

SonnetLab Netlist Tutorial

Specialists in High-Frequency Electromagnetic Software
(315) 453-3096 Fax: (315) 451-1694 http://www.sonnetsoftware.com

SonnetLab Netlist Tutorial

In this tutorial we will build a single network Sonnet netlist project. The circuit design for this tutorial will be arbitrary but this tutorial will use all of the availible netlist elements.

The .m file for this tutorial can be found in

```
< SonnetLab directory>\Tutorials\Netlist\Netlist.m
```

The first thing we do in this tutorial is create a new Sonnet project

```
Project=SonnetProject();
```

New Sonnet projects created with the above command will be Sonnet geometry projects; we want this project to be a Sonnet netlist project. SonnetLab makes it easy to convert Sonnet geometry projects into Sonnet netlist projects. This can be accomplished by using the following:

```
Project=initializeNetlist();
```

The above command will convert the Sonnet project into a default Sonnet netlist project that has the same settings as a Sonnet netlist project built using the Sonnet GUI. The initializeNetlist() method can be used on any Sonnet project in order to convert it to a default Sonnet netlist project.

In this tutorial we will create an arbirary circuit that uses several circuit elements. The Sonnet file format includes support for many different types of netlist elements including support for importing an entire Sonnet netlist project as a netlist element.

The first element that will be added to the circuit is a 50 unit resistor element between port one and two. The value for the units is defined globally for the project; the default unit selection is ohms. The resistor can be added to the circuit using the following command:

```
Project.addResistorElement(1,2,50);
```

The arguments for addResistorElement (first node, second node, resistence) is of the same form for capacitor elements (addCapacitorElement) and inductor elements (addInductorElement).

SonnetLab allows users to create a circuit element that is only connected to one node with the other node of the element floating. This can be accomplished by having the second argument be an empty matrix.

```
Project.addCapacitorElement(1,[],50);
```

The Sonnet netlist engine also has support for transmission lines and physical transmission lines. Transmission lines require the following arguments:

1. Node 1
2. Node 2
3. Impedence
4. Electrical Length

5. Frequency
6. (Optional) The Network Number

If we do not specify the optional argument for the element the element will be added to the first network in the project. A Transmission line connected between nodes 1 and 2 of the network, with an impedence of 100 units, an electrical length of 1000 units and a frequency value of 10 units can be added to the network with the following command:

```
Project.addTransmissionLineElement(1,2,100,1000,10);
```

The arguments for a physical transmission line are:

1. Node 1
2. Node 2
3. Impedence
4. Length
5. Frequency
6. EEFF
7. Attenuation
8. (Optional) The Network Number
9. (Optional) The Ground Node

In this tutorial we will add a physical transmission line connected from node 1 to 2 with an impedance of 100, a length of 1000, a frequency of 10, an eeff of 1, and an attenuation of 10.

```
Project.addPhysicalTransmissionLineElement(1,2,100,1000,10,1,10);
```

One of the greatest strengths of the Sonnet netlist engine is that it has support for loading touchstone files and Sonnet project files as netlist elments. These features allow users to incorporate many different design models in a network.

Users can add a touchstone file to a network using the following command:

```
Project.addDataResponseFileElement('data.s2p',[1,2]);
```

The above command will incorporate the touchstone file 'data.s2p' into the network at nodes one and two.

A Sonnet project file can be added to a Sonnet netlist using the addProjectFileElement() method. The addProjectFileElement() method requires the following arguments:

1. Filename
2. The nodes at which to place the project file element
3. Whether to use the project's frequency sweep or the subproject's frequency sweep

The third argument should be either a zero or a one; a value of zero indicates that the simulation should use the frequency sweep from the main project and a value of one indicates that the simulation should use the frequency sweep from the project file element. The following command is used in this tutorial to add the project file 'projectFile.son' as an element in our netlist.

```
Project.addProjectFileElement('projectFile.son',[1,2],0);
```

This concludes the adding elements phase of our tutorial. It is simple to add a new netlist element

to a circuit using the built in methods.

Before the netlist project can be simulated the frequency sweep settings must be set. In this tutorial we are going to use an ABS frequency sweep from 5 Ghz to 10 Ghz. This frequency sweep can be added to the project with the following command:

```
Project.addAbsFrequencySweep(5,10);
```

When a new project is created using SonnetLab it will need to be saved to the hard drive before it can be simulated. The first time a user wants to save a particular Sonnet project to the hard drive they must specify a filename for the project; this can be accomplished using the saveAs() method as follows:

```
Project.saveAs('Netlist.son');
```

The above command will save the Sonnet project specified by the variable Project to the hard drive with a file named 'SingleStub.son'. After the user does a single saveAs() command they are then able to use the save() command which will save the file to the hard drive using the same filename that was specified by the most recent call to saveAs(). If a Sonnet project file is opened from the hard drive (rather than being created from scratch from within Matlab) the user does not need to call saveAs() in order to specify a filename for the project file; the save() function will overwrite the read project file. At any time any Sonnet project may be saved with saveAs() in order to specify a different filename; the result of which will also cause all later calls of save() to be saved to the new filename.

Now that we have specified the simulation settings and have saved the project we can simulate the project using the command:

```
Project.simulate();
```

When the simulation is complete the user may examine the results using the Sonnet response viewer. SonnetLab can automate opening the Sonnet response viewer by issueing the following command:

```
Project.viewResponseData();
```