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

SonnetLab Multi-Network Netlist Tutorial

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

Version 1.1

Multi-Network Netlist Tutorial

In this tutorial will build upon the concepts shown in the single network netlist tutorial; it is recommended to read the single network tutorial first. This tutorial will show the user how to easy it is to manage a multi-network netlist project.

The .m file for this tutorial can be found in

```
< SonnetLab directory>\Tutorials\Multi-Network
Netlist\MultiNetworkNetlist.m
```

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

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

New Sonnet projects created with this command will be Sonnet geometry projects. In this tutorial we are going to make 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 two networks and 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.

When a new Sonnet netlist project is created the project will only have one element: a netlist network element. In this tutorial we will add a second network to the project. Users must specify the name of the new network, the port numbers for the network, and the resistence of the ports. The following command will add a network to the project named `'NetName1'` with four 50 unit impedence ports.

```
Project.addNetworkElement('NetName1',[1 2 3 4],50);
```

The methods used in the single network netlist tutorial are capable of being used to add network elements to any network in a Sonnet netlist project. The element creation methods all have optional parameters that give the user the ability to choose which network to add an element to. As seen in the single network tutorial if no value for the network is specified then the selected network will be the first network.

The first element that will be added to the circuit is a 50 unit resistor element between port one and two on network one. The value for the network number may be stated to be one or it maybe be omitted and achieve the same result:

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

We will next add an inductor element to node three and leave the second node empty. The inductor will be placed in the second network.

```
Project.addInductorElement(3,[],50,2);
```

The value for the network may either be the index for the network or the name of the network. In the above command the network index was used to add an inductor element to the second network of the project. In the following command the name for the network will be used to add a capactior element to the second network.

```
Project.addCapacitorElement(3,[],50,'NetName1');
```

The addPhysicalTransmissionLineElement() method has two optional arguments. The first optional argument is the network number/name. The second optional number is the ground reference node. The following command will add a physical transmission line element to the second network of the project. The transmission line will be 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. The transmission line will grounded at port 1.

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

The addDataResponseFileElement() method also has support for two optional arguments. The first one is the network number/name for the network the element resides in. The second optional argument specifies the ground node for the data response file element. The tutorial next will add the file 'Data.s2p' to the second network with a ground node at port four.

```
Project.addDataResponseFileElement('Data.s2p',[1,2],2,4);
```

This concludes the adding elements phase for our tutorial. The netlist creation methods that were used in the single network tutorial can easily be used in a multi-network scenario.

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('MultiNetworkNetlist.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();
```