

SimSmith N Block

(SimSmith version 9.4)

Version 9 of SimSmith introduces the 'N' block. Much like the 'F' block, the 'N' block provides an 'escape' capability; the 'F' block allows the user to write a 'Function' and the 'N' block allows the user to write a 'Netlist'.

This document presents the basics of creating a netlist; it is REALLY simple. It should be noted that the 'N' block is, as Brian Kernighan once said, "a VERY sharp scalpel with which the user CAN and WILL cut himself." Many things can go wrong in writing and evaluating a netlist and SimSmith provides very little insight when something DOES go wrong. The N block requires a certain amount of tolerance to computer (and my) stupidity.

Still, the 'N' block is extremely powerful and allows the description of a wide variety of circuits not previously possible in SimSmith.

Remember, SimSmith is, largely, CASE DEPENDENT.... CAPITALIZATION MATTERS!

In the following, `<*>` means a name. Names start with a letter and can have any number of letters and/or digits after the initial letter. Some names need to start with a particular letter: `<V*>` is a name which starts with a 'V', `<vcvs*>` is a name which starts with "vcvs", etc. If more than one name is needed, a digit replaces the *. For example, `<1>` is the 'first name', `<2>` is the second... etc.

(Names can refer to electrical nodes or to components. SimSmith does not try to make sure you are using reasonable names. For example, a resistor might be "Ra a b c;" which describes a resistor 'Ra' with value provided by the parameter variable 'a' between electrical nodes 'b' and 'c'.)

Also, in the following the term `<#>` is any number or expression which results in a number. The construct `<#dog>` indicates a number for the port named 'dog'. For example, "1+2" is a `<#>`, so is "a" where "a" will be

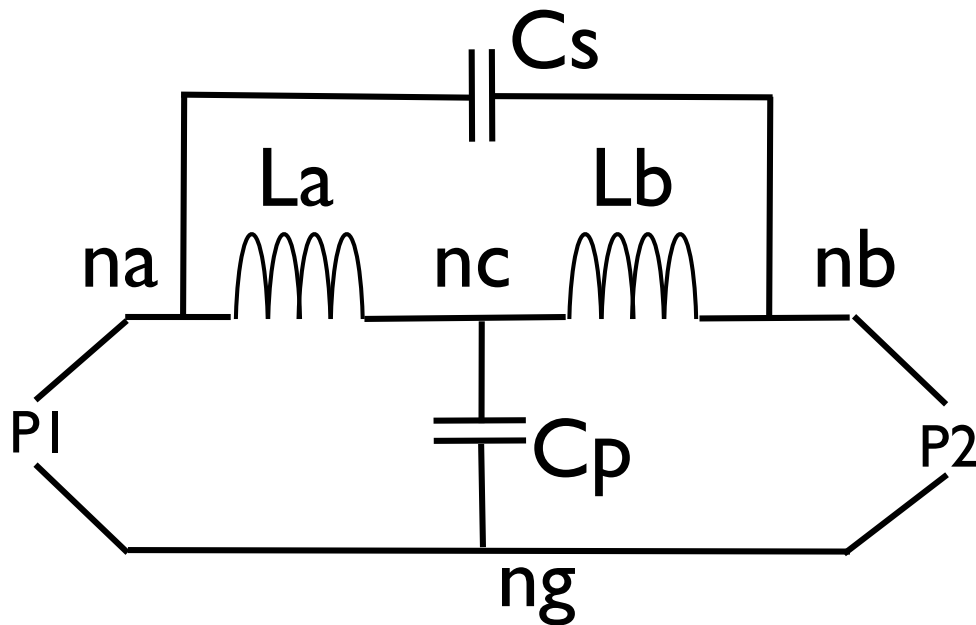
the value of the parameter. The expression syntax is taken from the 'F' block, see the "F Block Syntax" in the 'help' menu for more details.

The [] construct is use to indicate 'optional' arguments.

Finally, the 'N' block also allows for assignments the same as the 'F' block. Thus, one can say "\$a = 1+2;" and use \$a anywhere you need a number.

Example

To introduce most of the important data I use an example. Consider the following circuit. Note that this circuit can be implemented, without Cs, in SimSmith without using the N block. This makes it easy to compare the 'correctness' of the N block implementation.



The above circuit 'netlist' would be:

```
P1 ng na;          // port 1 is between nodes ng and na
P2 ng nb;          // port 2 is between nodes ng and nb
La 795n na nc;     // La is a 795nH inductor between nodes na and nc
Lb 795n nb nc;     // see above.
Cp 318p ng nc;     // Cp is a 318pF capacitor between nodes ng and nc
Cs 10p na nb;      // Cs is a 10p capacitor between nodes na and nb
```

All commands end with a ';'. There may be multiple commands on a line. The "//" is a comment to the end of the line. Otherwise, whitespace is largely ignored.

Taking the cue from Spice, all commands start with a designator. The first letter of the designator is the 'type' of component; L for inductor, C for capacitor, P for port, R for resistor and 'X' for reactance, etc.

The 'Px' commands are unique in that they have no 'value', only the node names between which the ports are defined. Note that "P1 a b;" is NOT the same as "P1 b a;"

The 'R', 'X', 'L', and 'C' commands take a value followed by two node names. The 'R' is a resistance, 'X' reactance, 'L' inductance, and 'C' capacitance. There are no enforced restrictions on the values. For example, the commands "Ra j1 a b;" is the same as "X 1 a b;". Negative values are allowed as well. Indeed, to really confuse things,

```
Ra L400n a b;      // same as 400n inductor
La 50/jw a b;      // same as 50 ohm resistor.
```

COMMANDS

Here are the actual command syntaxes...

INDEPENDENT DEVICES:

P1 <1> <2>;
P2 <1> <2>;

<R*> <#Ohms> <1> <2>; // resistor

<X*> <#jOhms> <1> <2>; // reactance.

<L*> <#H> <1> <2> [<#Q [#FofQ]]; // inductance optional Q & FofQ

<C*> <#F> <1> <2> [<#Q [#FofQ]]; // capacitance optional Q & FofQ

<V*> <#Volts> <1> <2>; // voltage.

<I*> <#Amperes> <1> <2>; // current

CONTROLLED SOURCES:

Current Controlled Current Source.

Sense current going through component <*> and generate current from <1> to <2> with gain #.

```
<cccs*>    <1> <2> <*>  #;
```

Current Controlled Voltage Source.

Sense current going through component <*> and generate voltage from <1> to <2> with gain #.

```
<ccvs*>    <1> <2> <*>  #;
```

Voltage Controlled Voltage Source.

Sense voltage <3> to <4> and generate voltage from <1> to <2> with gain #.

```
<vcvs*>    <1> <2> <3> <4> #;
```

Voltage Controlled Current Source.

Sense voltage <3> to <4> and generate current from <1> to <2> with gain #.

```
<vccs*>    <1> <2> <3> <4> #;
```

For the new comer to this stuff, current sources generate current from <1> to <2> which means current is sucked into <1> and sent out <2>. This confused me for quite some time when trying to debug various circuits. Just a 'heads up'!

TRANSMISSION LINE:

Of course, no Smith chart program would be complete without transmission line support: #1 is length. One end is the port <1><2>, the other end is <3><4>.

```
<trans*> <1> <2> <3> <4> <#len> [meters]  
        [<#Zo> [<#VF> [<#lossPerHundred> [<#losFrequency>]]]] ;
```

Note that this transmission line is theoretical and allows one end to 'float' with respect to the other. This can cause Matrix Inversion problems if you forget to restrict the voltages at one end. This doesn't happen often, most of the time ports <2> and <4> will be connected, for example, but this is YOUR choice and Modified Nodal Analysis may blow up if you leave the voltages unrestricted.

Even for a zero length transmission line, there is no connection between <1> and <2> or <3> and <4>! (Note, a zero length transmission line might be lengthened to a very short line in order to avoid numerical problems.)

TRANSFORMER:

And to more or less fill the menu, one needs a transformer. To be consistent with SimSmith and with Spice, transformers are modeled as coupled inductors. Taking the cue from Spice, SimSmith leaves the inductors alone and provides for a 'coupling' coefficient. The command is:

```
<K*> <1> <2> <#k> ;    // couple device <1> to <2> with factor k.
```

Just a quick note, SimSmith does not ensure the <1><2> above are inductors. Indeed, you can use capacitors or resistors or even make them different types; sharp scalpel, be careful!

Coupling of more than 2 devices is not yet supported.

Sinclair Monofilament:

Finally, for the truly ambitious, the SimSmith Netlist block allows you to make your own entries in the matrix. To understand how this is used, the reader is encouraged to go bone up on “Modified Nodal Analysis” in general and matrix “stamps” in particular.

Assuming a certain familiarity with these ideas the “stamp” command will make sense. For the moment: when you add a stamp to the matrix you generally have to provide entries for each ‘row’ and ‘column’ and an entry in the ‘result’ The format of the equation statement is:

$$\text{<stamp*> <1> <\#row>,<\#col> [<2><\#row>,<\#col>[....]] = <\#result>;}$$

For example, a simple 3 volt voltage source between a and b would be:

stamp0 a 1,1 b -1,-1 = 3;

Honestly, I don’t expect anyone to use this but... it is how I confirmed what the matrix entries would be for each of the components before writing the java code to ‘build it in’. Since its there, I figured I’d tell the user about it.

BACKGROUND

SimSmith processes the netlist using a methodology called “Modified Nodal Analysis”. In essence, it writes an equation for the current leaving each of the nodes. It then solves this set of equations using numeric, Gaussian elimination and backward substitution.

The specifics can be found in “Electronic circuit & System Simulation Methods” by T. L. Pillage et. al. The techniques used in SimSmith are contained entirely within the first 40 pages of this nearly 400 page book.

In the end, the whole technique relies on the ability numerically to solve the many simultaneous equations and this requires the ‘inversion’ of a matrix. Not all matrices are ‘invertable’ and when one is not, SimSmith simply throws up its hands and says, “matrix not invertable” or some such. Further, not all matrices that are ‘theoretically invertable’ can be inverted using present day floating point implementations.

There are numerous (innumerable in fact) things that can make my simple implementation break down. For hints, read any text on numerical analysis that deals with the issues of finite arithmetic precision. It’s a nasty business, floating point arithmetic, and while I implement some precautions, I’m a neophyte and this is just a hobby.

Ultimately, the ‘N’ block can be extremely useful if the user is willing venture in without a net, and so, I put in the N block for the advanced user. Please understand, I know it can be broken and I know SimSmith won’t warn you or explain why it fails. I am very interested in common practices that can cause failure; not so interested in deliberate attempts to cause failure.

There are also some annoying things that occur when you are editing a netlist AND changing the parameters of the block. I know about these and fix them when time and interest allow. I truly appreciate your forbearance on these issues.

Wrap UP

With the exception of adding the occasional component, I don't expect to make significant improvements to the 'numerical analysis' subsystem in SimSmith; Spice is a better answer if the circuit is too complex or constantly fails to evaluate. The N block will suffer from this decision, BUT, there are very real things you can do with the 'N' block and I hope giving you access to it will help in your endeavors.

Ward