

***SCREAMER,***

***A Pulsed Power Design Tool***

**User’s Guide for Version 4.4.3**

**Written by:**

Rick B. Spielman Mark L. Kiefer Kelley L. Shaw Ken W. Struve Mel M. Widner

## October 20, 2020

Copyright 1996 Sandia Corporation. Screamer is an open-source computer code that is released under a GNU license.

**Contents**

1. [Background 6](#_TOC_250093)
2. [The Model 7](#_TOC_250092)
3. Defining a SCREAMER Problem 11
   1. SCREAMER Circuit Blocks 11
      1. [Transmission Line Circuit Blocks 11](#_TOC_250091)
      2. [General -Section Block and -Section Block Subsets 13](#_TOC_250090)
      3. [Source Blocks 14](#_TOC_250089)
   2. [Initial Conditions 15](#_TOC_250088)
   3. [Variable Circuit Elements 15](#_TOC_250087)
      1. [Variable Element Models Contained in the SCREAMER “Library” 15](#_TOC_250086)
      2. [Switched Variable Element Models Contained in the SCREAMER “Library” 33](#_TOC_250085)
4. Input/Output Overview of SCREAMER 35
   1. Input to SCREAMER 35
      1. [Run-Time Options 35](#_TOC_250084)
      2. The SCREAMER Input File 35
      3. [User Subroutines 35](#_TOC_250083)
   2. Output from SCREAMER 35
      1. The SCREAMER Log File 35
      2. [Circuit Parameter Output 36](#_TOC_250082)
5. [Constructing an Input File 38](#_TOC_250081)
   1. [Conventions 38](#_TOC_250080)
   2. [Setup Conditions 38](#_TOC_250079)
   3. [Circuit Description 40](#_TOC_250078)
      1. [Circuit Branch Locations 40](#_TOC_250077)
      2. [Circuit Blocks 42](#_TOC_250076)
      3. [Initial Conditions 46](#_TOC_250075)
      4. [Variable Elements 46](#_TOC_250074)
   4. [Output Requests 54](#_TOC_250073)
   5. [Additional Output Requests for Models of Variable Elements 57](#_TOC_250072)
      1. [Cylindrical and spherical foil models 58](#_TOC_250071)
      2. [Gas-puff model, DPF, and Dynamic Hohlraum models 58](#_TOC_250070)
      3. [Lossy switch model 58](#_TOC_250069)
      4. Zflow POS model 58
      5. [MFI model 58](#_TOC_250068)
      6. [Radiation Yields for the Cylindrical Foil Model 58](#_TOC_250067)
      7. Zflow Loss Model 59
      8. [Multiple Collapsing Shell (NSHELL) Model 59](#_TOC_250066)
   6. [Comments, Labels, and Titles 59](#_TOC_250065)
6. [Input File Summary 62](#_TOC_250064)
7. [Example Input Files 67](#_TOC_250063)
   * 1. [A Simple Capacitor Discharge Circuit 67](#_TOC_250062)
     2. [A Dynamic Hohlraum Circuit 68](#_TOC_250061)
     3. [The Marx Circuit 69](#_TOC_250060)
     4. [The Demon Model 70](#_TOC_250059)
     5. [PBFA II Convolute Model 72](#_TOC_250058)
     6. [Complex Branches 75](#_TOC_250057)
     7. [E-Beam Diode 77](#_TOC_250056)
8. [User Subroutines 80](#_TOC_250055)
   1. [Purpose of User Subroutines 80](#_TOC_250054)
   2. [Format of User Subroutines 80](#_TOC_250053)
   3. [Instructing SCREAMER to Use Your Subroutines 82](#_TOC_250052)
   4. [User Variables 82](#_TOC_250051)
      1. [User Variables Defined 82](#_TOC_250050)
      2. [Modifying your User Subroutines 82](#_TOC_250049)
      3. [Labels for User Variables 84](#_TOC_250048)
   5. [Example User Subroutines 84](#_TOC_250047)
      1. [Gas Switch Model 84](#_TOC_250046)
9. SCREAMER Output 86
   1. The SCREAMER Log File 86
      1. [Setup Parameter Summary 86](#_TOC_250045)
      2. [Listing of User Circuit 86](#_TOC_250044)
      3. [The Initial Circuit Status 87](#_TOC_250043)
      4. [Circuit Status at Specified Intervals 87](#_TOC_250042)
      5. [Execution Summary 88](#_TOC_250041)
   2. [Circuit Parameter Output 88](#_TOC_250040)
      1. [FILE 89](#_TOC_250039)
      2. [TABLE 89](#_TOC_250038)
      3. [TXT 90](#_TOC_250037)
      4. [PFF 91](#_TOC_250036)
      5. [CSV 91](#_TOC_250035)
      6. [SFC 91](#_TOC_250034)
   3. [Interactive Input/Output 92](#_TOC_250033)
   4. [File Naming Conventions in SCREAMER 94](#_TOC_250032)
10. [Installation and System Dependent Information 96](#_TOC_250031)
    1. [LINUX (and UNIX) 96](#_TOC_250030)
       1. [Installation 96](#_TOC_250029)
       2. [Obtaining Sample Input Files and Sample User Subroutines 96](#_TOC_250028)
       3. [Execution 97](#_TOC_250027)
    2. [Macintosh OS 97](#_TOC_250026)
       1. [Installation 97](#_TOC_250025)
       2. [Execution 97](#_TOC_250024)
       3. [Output 98](#_TOC_250023)
       4. [User-written Subroutines 98](#_TOC_250022)
    3. [Windows PC OS 98](#_TOC_250021)
       1. [Installation 98](#_TOC_250020)
       2. [Execution: 99](#_TOC_250019)
       3. [Output 99](#_TOC_250018)
11. OpenMP SCREAMER 100
12. [SCREAMER Circuit Parameters and Memory Limits 102](#_TOC_250017)
13. [Debugging Capabilities 103](#_TOC_250016)
14. [Problem Size and Run Time 103](#_TOC_250015)
    1. [Practical Time Step and Resolution Rules 104](#_TOC_250014)
15. [Versions 105](#_TOC_250013)
    1. [Source file names 105](#_TOC_250012)
    2. [Execution status 105](#_TOC_250011)
    3. [Modifications to circuit blocks and models 105](#_TOC_250010)
       1. [The Lossless and Lossy (MITL) Transmission Line Blocks 105](#_TOC_250009)
       2. [Racetrack MITL model 106](#_TOC_250008)
       3. [Perveance MITL model 106](#_TOC_250007)
16. [Compiling Your Own Version of Screamer 107](#_TOC_250006)
    1. [Install the GNU Compiler Collection (GCC) and gFortran 107](#_TOC_250005)
       1. [Example install 107](#_TOC_250004)
    2. [For the Macintosh – Install Xcode command line tools 107](#_TOC_250003)
    3. [Copy the latest version of SCREAMER to your Computer 107](#_TOC_250002)
    4. Compiling and Linking SCREAMER 108
    5. [Running Your New Screamer Version 109](#_TOC_250001)
    6. SCREAMER Change Control 109

[Appendix Tips and Tricks 111](#_TOC_250000)

**Figures**

Figure 1 General SCREAMER Circuit 2-1

Figure 2 SCREAMER Branch Types 2-2

Figure 3 Lossless Transmission Line Block 3-1

Figure 4 MITL Block 3-2

Figure 5 -Section Block and Subsets 3-3

Figure 6 SCREAMER Source Blocks 3-4

Figure 7 Hysteresis Curve used for Saturable Inductor Model 3-6

Figure 8 The *Zflow* POS model 3-11

Figure 9 Input/Output Flow for SCREAMER 4-3

Figure 10 Some Plotting Conventions for Circuit Blocks 5-14

Figure 11 Plotting Conventions for Circuit Source Blocks 5-15

Figure 12 Block Circuit Diagram of Marx Module 5-20

Figure 13 Plot of the correction factor for impedances between 1 and 100 Ohms .... A-2

# Background

SCREAMER is a special purpose circuit code developed as a design tool for single module accelerators. It is accurate, flexible, and user-friendly and, because of topological limitations for circuit configurations, it can efficiently simulate large circuits with very large numbers of nodes and circuit elements. Screamer is written in Fortran 77 with a few Fortran 90 extensions.

SCREAMER has been ported to GNU gFortran to maximize the cross platform compatibility. SCREAMER is now fully compatible with the gcc/gFortran open source compiler standards as of SCREAMER Version 3.2.7.

SCREAMER V4.4 and beyond is fully parallelized using OpenMP. The number of threads is user selectable.

SCREAMER is fully Open Source under the GNU license with user improvements and enhancement encouraged. Please send requests for the latest version of the source code to: [spierick@isu.edu](mailto:spierick@isu.edu) .

# The Model

The SCREAMER code solves energy transmission problems in series elements. It is one time level, fully implicit, and second-order accurate. It has the capability to handle variable circuit elements by selecting from various models which have been placed into the code or by inserting user-written Fortran subroutines. The circuit set-up is straightforward, fast, and user- friendly. Many forms of output files are available for post-analysis of circuit parameters and new output file types are easily added.



*C*

*i+1*

*G*

*i+1*

*C*

*i*

*Gi*

*Li*

*Ri*

*i+1*

*V*

*i+1*

*Ii*

*Vi*

*I*

*B*

*I*

*Bi*

Fig. 1. General SCREAMER Circuit

The general circuit element in **Fig. 1** is shown as the section in the dashed box in as part of a general -section. It consists of a shunt resistor, a shunt capacitor, a series resistor, and a series inductor all associated with a single circuit node. This is the basic circuit element and all user specified designs will be mapped to groups made from this element.

An external branch can be connected to any node or pair of adjacent nodes. The branch types are shown in **Fig. 2**. Top (series) and end (parallel) branches from the main branch (Level 1) may not reconnect to the main branch. Branches (Level 2) may contain branches (Level 3)! This topological limitation greatly simplifies and speeds the solution.

Fig. 2. SCREAMER Branch Types



End Branch

Main Branch

***•••***

***•••***

Top

Branch

For each node *i* in each branch, SCREAMER simply calculates the voltage drop between nodes and sets the sum of the currents entering a node to zero as shown in the following equations.

*Vi* – *Vi+1*

= *Ri Ii* +  *Li Ii*



*t*

(1 )

*Ii-1* – *Ii* – *IB*

*i*

= *Gi Vi* +  *Ci Vi*



*t*

(2 )

The finite difference representation is given by the following equations where *R*, *G*, *C*, and *L* are constant. Two time levels are considered: the new time (superscript *n*), for which the voltages and currents are not known, and the old time (superscript *o*), for which the voltages and currents are known. In this representation, the time step, *t*, is *tn – t0*. By evaluating the equations at the half time step, second order accuracy is obtained.

*Vn Vo V* = -----------------

+

*2*

*In Io*

+

*I* = ------------**-**

*2*

(3 )

(4 )

*n o*

 *CV* =

*t*

*C**V*

 *t*

= *CV* – *V* , where C is constant (5 )

Δ

*t*

*n o*

 *LI* =

*t*

*L**I* =

 *t*

*LI* – *I* , where L is constant (6 )

Δ

*t*

By substituting these into Eqs. 1 and 2, we get the result:

*Vn* – *Vn R L n*

*Vo* – *Vo R L o*

----*i*------------*i*-*+*---*1*- –  ----*i* + *i*  *I*

= ----*i*--*+*--*1*------------*i*- +  ----*i i*  *I*

(7 )

*2*  *2* Δ*t* *i*

*2*  *2* Δ*t* *i*

*In* – *I* – *I G C Io* + *Io* – *I G C*

*n n*

*o*

---*i*-*-*-*1*--------**-***i*-----------*B***-***i* –  ---**-***i* + ---**-***i* *Vn* = ---*i*----------*B*---*i*---------*i*-*-*--*1* +  ---**-***i* *i* *Vo*

(8 )

*2*  *2*

Δ*t* *i*

*2*  *2*

Δ*t* *i*

At time *t = 0*, all component values and the initial conditions for all the currents and voltages are specified. Generally, you only need to specify a small portion of the component values and identify a few sources to initialize the circuit. Since adjacent elements are coupled, a system of linear equations must be solved simultaneously to obtain the voltages and currents at the new time. This requires inversion of a block tridiagonal matrix with a few off-diagonal elements if there are top and end branches attached to the main branch.

This formulation allows *R*, *G*, *C*, and *L* to be variable elements. You can specify a variable element by choosing one from the SCREAMER model library or by defining one yourself with a Fortran subroutine. A procedure for accommodating variable *R*, *G*, *C*, and *L* terms is shown below. To utilize this, *Li* and *Ci* are first replaced by *∂Li/∂Ii* and *∂CiVi/∂Vi* in Equations 7 and 8. Note that this assumes *L* depends only on the current passing through it and *C*, the voltage across it. *R* and *G* have no restrictions on their dependencies. In essence, SCREAMER replaces *R*, *G*, *∂CV/∂V*, and *∂LI/∂I* with their value at the half time step, *th = (tn + t0)/2*.

*R* = *f* *Ih Vh Vh*

 *t* 

(9 )

*i i*  *i*  *i+1*

*h*

 *L I*

= *f* *Ih*

*Ii i i i*

*G* = *f* *Vh Ih Ih*  *Ih*  *th*

(10 )

(11 )

*i i*  *i*  *i-1 Bi*

 *C V*

= *f* *Vh*

*Vi i i i*

(12 )

Here, *f* is the function describing the behavior of the variable element and *Ih* and *Vh* are the current and voltage at *t* = *th*. Of course, the expressions on the right-hand side are not known at *t* = *th*, unless *R* and *G* depend only upon the time. Typically, values at the previous half-time step, *th – t*, are used instead.

For a single line, the procedure for solving for all *Vni* and *Ini* is straightforward and one simply inverts a block tridiagonal matrix. Note that the number of calculations in this procedure scales linearly in the number of circuit nodes, while a full matrix solution scales as the number of nodes squared. Complications arise when one adds branches as shown in **Fig. 2**. This leads to a few off-diagonal rows and columns. SCREAMER takes advantage of the sparseness of this matrix, resulting in an efficient algorithm which still scales nearly linearly in the number of nodes.

1. **Defining a SCREAMER Problem**

To define a SCREAMER problem, you must first identify the main, secondary, and tertiary branches. (See **Fig. 2**.) These branches are series arrangements of circuit elements as shown in **Fig. 1**. To define a branch, you simply specify circuit blocks connected in series. This is a compact method of identifying circuit parameters. Blocks in a secondary branch are ordered from the point they exit the main branch to their ends. This can appear backward.

* 1. **SCREAMER Circuit Blocks**

SCREAMER provides three categories of circuit blocks. They are (1) transmission-line circuit blocks, (2) general -section blocks and its subsets, and (3) source blocks.

### Transmission Line Circuit Blocks

* + - 1. Lossless Transmission Line

The lossless transmission line consists of series inductor and shunt capacitor elements with no resistance. To characterize a transmission line, you specify a transmission line length in time,

**, the input and output impedances, *Zin* and *Zout*, and a resolution time, *tres*. Based on this resolution time parameter, SCREAMER will break the line into segments with delay times of

approximately *tres* as shown in **Fig 3**. The impedance may vary over the length of the transmission line in a linear or exponential fashion and SCREAMER uses the interpolated impedance and the fractional time delay when calculating an individual segment’s *L* and *C* values.





*res*

Δ ***=*** *t*

*res*

***Zout***

***•••***

Δ ***=*** *t*

Δ ***=*** *tres*

***Zin***

Fig. 3. Lossless Transmission Line Block

For each segment,

** = *LC*

and

*Zsegment* =

* + - 1. Lossy Transmission Lines



*L C*

The lossy transmission line consists of series inductors & resistors and shunt capacitors & resistors. The resistor values are fixed by the user. To characterize a line, you specify a TL

length in time, **, the input & output impedances, *Zin* & *Zout*, the total shunt resistance *Rshunt*,

the total series resistance, *Rseries*, and a resolution time, *tres*. Based on this resolution time parameter, SCREAMER will break the total length of the transmission line into segments with delay times of approximately *tres*. The impedance may vary over the lossy line only in a linear fashion and SCREAMER uses the interpolated impedance and the fractional line length when calculating an individual segment’s *L* & *C* values and *R2* and *G1* values. An example of a lossy transmission line is shown in **Fig. 4**.

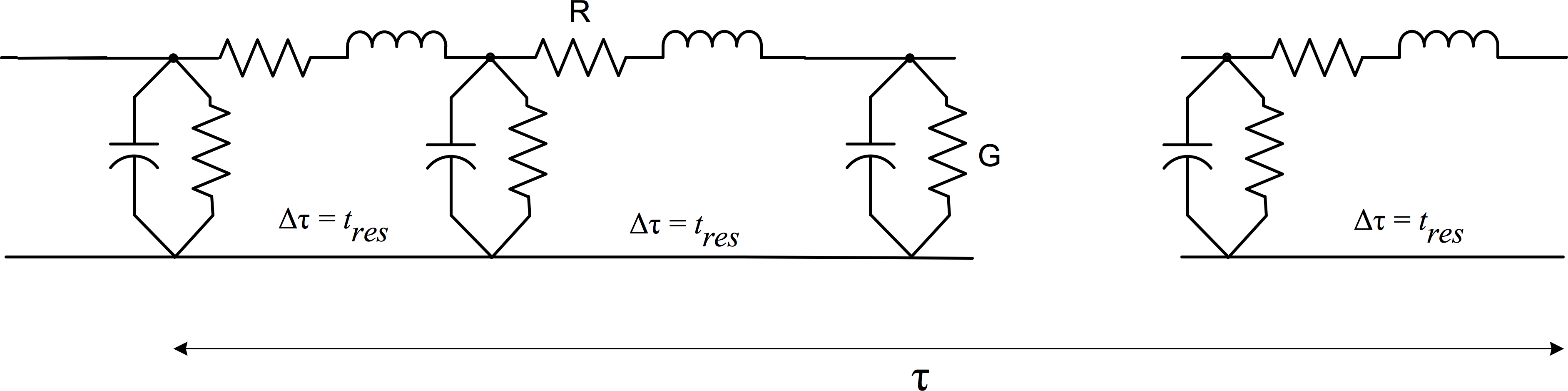


Fig. 4. Lossy Transmission Line Blocks

* + - 1. Lossy (Magnetically Insulated) Transmission Lines

Lossy transmission lines are achieved with two types of MITL blocks. The first is a racetrack- type MITL. This has the same basic structure as a lossless transmission line and is broken up

into segments with delay times of *tres* also, but adds variable shunt resistors to the individual circuit segments as shown in **Fig. 5**. This model has been found to agree well with 2D particle- in-cell simulations. You specify the gap and circumference of the MITL (both assumed to be constant or nearly so). The assumption is that the electric and magnetic fields are nearly constant across the gap. This is true if the gap is small relative to the circumference. For example, one could use a coaxial MITL. Here, the gap would be the distance:

*g = r o u t e r – r i n n e r << r o u t e r*

and the circumference would be:

*c = 2  r o u t e r  2  r i n n e r*

This model utilizes some time-averaging of circuit parameters in order to increase its numerical stability; however, it may show numerical oscillations if the time-step is not small enough. In

general, *tres* should be chosen to be at least ten times smaller than the fastest current variation expected. In addition, the time step, *t*, should be chosen to be two to five times smaller than *tres*. If oscillations are observed, reduce *t* AND *tres* by a factor of two and try again. It is also wise to test any answer by running with a smaller time step to verify a correct solution.

Finally, there is an optional model parameter for specifying the electric field at which cathode emission is turned on. If not specified, emission is not allowed until the electric field reaches 200 kV/cm (*2X107* volts/m).



***•••***

*, Z*

Fig. 5. MITL Block

The second type of MITL block is based on a perveance description and uses no geometrical parameters at the expense of a preliminary user calculation of the line’s perveance. In MKS units, the perveance is typically calculated from the formula

*P* = 2.33*X* 106 2

*G dA*

where *G* is the gap and *A* is the area of the smallest electrode. An emission turn-on electric field, identical to that used in the racetrack MITL model (although it cannot be set by the user) is available. To determine this value of the electric field, the model infers a transmission line gap from the perveance and line impedance. A coaxial transmission line and a constant gap are assumed in order to do this calculation.

This model does not as closely reproduce 2D particle-in-cell simulation results as the "racetrack" MITL model. In general, it predicts more loss at lower currents and less loss at higher currents. However, the total energy loss is not significantly different between the two MITL models.

### General -Section Block and -Section Block Subsets

**Figure 6** shows a general -section block and three subsets of it which are included for convenience. For the -section, all six circuit components must be specified. The *RC* to ground block, consisting of a shunt resistor and capacitor, requires only one node. The *RL* series block, consisting of a series resistor and inductor, requires two nodes. The adder block, consisting of a 1-MΩ series resistor, also requires two nodes. The adder is useful for connecting a top branch to the main branch so that the voltage of the top branch is added to that of the main branch.

For most pulsed power designs, a resistance of 1 MΩ will adequately describe an open circuit and 1 µΩ will suffice for a short circuit.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *R*  *1* |  | *R L*  *2 2*  *C R*  *1 3*  *Section* |  | *C3* | *R2 L*  *2*  *RL in Series* |
| *R C*  *1 1*  *RC to Ground* | | | | | *1M*  *Adder* |

Fig. 6. -Section Block and Subsets



Please note that the numeric designation for resistors, capacitors, and inductors shown in **Fig. 6** are not arbitrary and are used to set variable parameters, set initial values, etc. It is handy to remember the resistor and capacitor designations.

### Source Blocks

In **Fig. 7**, the four source blocks are shown. The first is the voltage source block which may only occur as the first block in the main branch. The second is the end-of-branch voltage source and may only occur as the last block in a main or secondary branch. The third is the current source block and it also may only occur as the first block in the main branch. The last is the end-of-branch current source and may only occur as the last block in a main or secondary branch.

Note well the polarity of the applied voltage if you use an end-of-branch voltage source or the direction of the current flow if you use an end-of-branch current source on secondary branches. **Fig. 1** and **Fig. 2** show the conventions for voltage polarity and current direction.



|  |  |
| --- | --- |
| **R2 L2**  **V**  **Beginning-of-Main-Branch Voltage Source** | **R2 L2 V**  **End-of-Branch Voltage Source** |
| **I**  **R3 C3**  **Beginning-of-Main-Branch Current Source** | **I**  **R1 C1**  **End-of-Branch Current Source** |

Fig. 7. SCREAMER Source Blocks



## Initial Conditions

In addition to the source blocks described in the previous section, energy may be placed in the circuit by setting an initial voltage on a capacitor or on a lossless transmission line or by placing an initial current in an inductor or in a lossless transmission line.

## Variable Circuit Elements

SCREAMER provides you with the capability of specifying variable elements.

### Variable Element Models Contained in the SCREAMER “Library”

Currently, the SCREAMER model "library" contains the following variable element models. Additional documentation about these models is in **Section 5.3**.

* + - 1. Exponential Switch Model

The exponential switch model (gas-switch model) is used to describe the behavior of a resistor using the following equations, where *R(t)* is the resistance at time *t*:

*R( t ) = Rope n* , when ( *t < tsw i t c h* )

*R**t* =

*Zswitche*–

*Rclose* + , when

*t*  *tswitch* 

*1* – *e*– + *Z*

*switch*

*10*–*6*

* = ( t – t s w i t c h )/ *

*R o p e n* , *R c l o s e* , *t s w i t c h* , **  and *Zs w i t c h* ar e al l us er spe cif i ed .

* + - 1. Decay Switch Model

The decay switch model is used to describe a closing switch. The resistance remains constant at *Ropen* until the time specified by *tswitch*. At that time, the value of the resistance decays

exponentially to *Rclose* as described by the following equations, where *R(t)* is the resistance at time *t*:

*R( t ) = R o p e n* , when ( *t < t s w i t c h* )

*R( t ) = R + ( R – R )e- * , when ( *t > t* )

*c l o s e o p e n c l o s e s w i t c h*

* = ( t – t s w i t c h )/ *

*Ropen*, *Rclose*, *tswitch*, and ** are all user specified.

* + - 1. Rise Switch Model

The rise switch model is used to describe an opening switch. The resistance remains constant at *Rclose* until the time specified by *tswitch*. At that time, the value of the resistance rises exponentially to *Ropen* as described by the following equations, where *R(t)* is the resistance at time *t*:

*R( t ) = R c l o s e* , when ( *t < t s w i t c h* )

*R( t ) = R + ( R – R )(1 - e -  )* , when *(t > t )*

*c l o s e o p e n c l o s e s w i t c h*

* = ( t – t s w i t c h )/ *

*Ropen*, *Rclose*, *tswitch*, and ** are all user specified.

* + - 1. Magnetic Switch Model

The magnetic switch model is used to describe a saturable core inductor. **Fig. 8** shows the hysteresis curve used for the model. *H1*, *Hsat*, *Hrev*, and *Bsat* determine the relative permeability of the switch core in its various states of saturation. The values that must be provided by the user

are *PF, Ri*, *Ro*, *W*, *H1*, *Hsat*, *Hrev*, and *Bsat*. *PF* is the core packing fraction, *Ri* is the inner radius of the switch, *Ro* is the outer radius of the switch, and *W* is the switch width.

*Bsat*

*B*

*Hrev*

*H*

*H1 Hsat*

Fig. 8. Hysteresis Curve used for Saturable Inductor Model

* + - 1. Time-Switch Plasma Opening Switch Model

The time-switch plasma opening switch model is used to describe a shunt resistor. The resistance of the switch is given by *Rmin*, until *tswitch*, at which time it begins to open. After the switch opens, the resistance is determined by the following equation:

*R* =  *Q* 

*min*  *Rmax*



*V*

*K*

where *Q* is the charge into and past the switch since *tswitch* occurred, and *V* is the voltage across the switch. *tswitch*, *K*, *Rmax*, and *Rmin* are all user specified.

* + - 1. Charge-Switch Plasma Opening Switch

The charge-switch plasma opening switch model is used to describe a shunt resistor. The resistance of the switch is given by *Rmin*, until the accumulated charge passing through the switch reaches *Qswitch*, at which point it begins to open. After the switch opens, the resistance is determined by the following equation:

*R* =  *Q* 

*min*  *Rmax*

 

*K V*

where *Q* is the charge into and past the switch since *Qswitch* occurred, and *V* is the voltage across the switch. *Qswitch, K*, *Rmax*, and *Rmin* are all user specified.

* + - 1. The Diode Model (Classical Electrical Diode)

The classical electrical diode model is uses a variable shunt resistor acting as an classic electrical diode. The resistance (conductance) of the diode is given by providing six (6) voltage and current pairs that describe the diode with a linear fit between the pairs of points. The pairs of points are: V1, I1 V2, I2 V3, I3 – the pairs specifying the reverse holdoff parameters. The point at 0, 0 is assumed so that all diode models pass through the origin of the diode curve. The pairs V4, I4 V5, I5 V6, I6 describe the diode in forward conduction. Figure 9 below shows a typical diode curve.

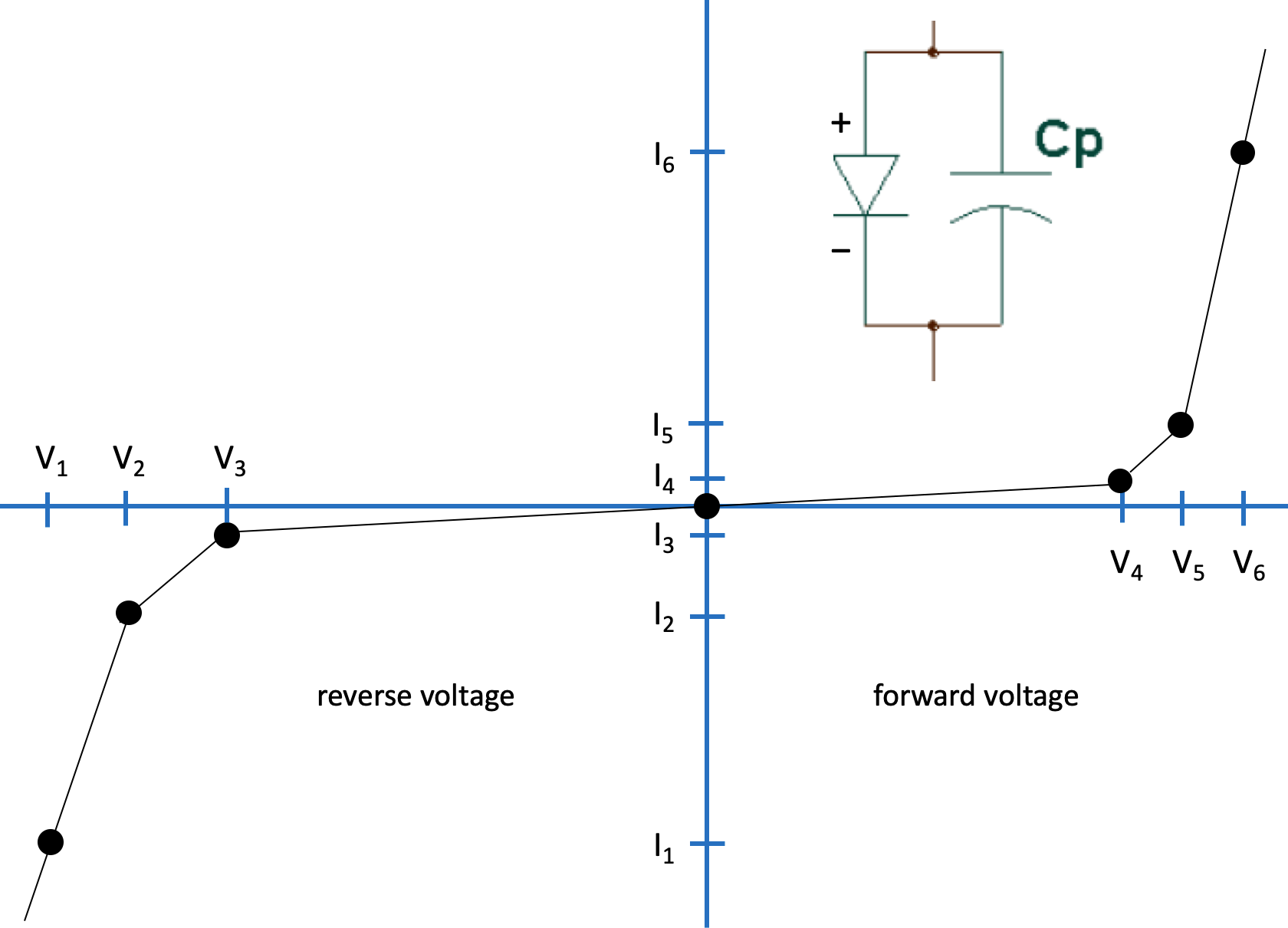


Fig. 9. Diode V-I curve used for the diode model.

From Equation 8, we see that SCREAMER actually uses the conductance of the diode, *1/Rc*, rather than its resistance. The Diode Model is only available as a RCGround and an PI element. Using a diode in series with a branch is easily done using a top branch.

* + - 1. The Slutz Diode Model (Ion Diode)

The Slutz diode model is used to describe a shunt resistor acting as an ion diode. The resistance of the diode is given by the following algorithm, where *V* is the voltage across the diode, *td* is the time delay before beginning gap closure, *Rmax* and *Rmin* are the maximum and minimum resistances the diode may have for positive *V*, *A* is the effective diode area, *G* in the initial gap,

*v* is the gap closure velocity, and *Gmin* the minimum gap allowed. *Pmratio* specifies the ratio of the proton mass to the mass of the ion produced by the diode:

*R = 1 X 106* , when *V < 0*

*R = Rm a x* , when *V > 0* , *t < t d*

*R = min( Rm a x ,max( R m i n ,Rc ))* , when *V > 0* , *t > t d*

where

*Rc* =



*G2*

*5.5*10–*8A*

*e f*-*f*

*1*

*V*

*Pmratio*

and

*Ge f f = max( Gm i n ,G- v( t – t d ))*

From Equation 8, we see that SCREAMER actually uses the conductance of the diode, *1/Rc*, rather than its resistance. Thus, *Gmin* is needed to avoid division by zero. *td*, *Rmax*, *Rmin*, *A*, *G*, *v*, *Gmin*, and *Pmratio* are all user specified.

* + - 1. The Cylindrical Foil Model.

This model comprises an entire custom circuit block NOT a variable element in a standard RLseries block. It provides diagnostics of foil velocity, foil position, kinetic energy, and other electrical parameters. The model calculates and includes in the circuit the changing inductance of current-driven collapsing foil or wire array. This effective dissipation is measured by

*R* = ---*L*-**-**

*t* .

The inductance of the dynamic load, *L,* is the calculated inductance determined by the position of the foil. Model input parameters are the foil initial and final radius, the foil length, and the foil mass. The model calculates the magnetic acceleration of the foil as

*at* =

*2*

*0 t-1*

 *I l*

*2mrt-1* ,

where *r* is the foil radius*, l* is the foil length, *m* is the foil mass, and *I* is the instantaneous foil current (time step by time step). The velocity and radius at time *t* are:

*vt* =

*vt-1* + *at* Δ*t* and

*rt* =

*rt-1*

*v* + *v*

---------------**-** Δ*t*

+ *t t-1*

*2* .

The inductance and time-derivative of the inductance are:

*L* = 0 *l* ln

, and

*ro*

*ri*

= *Lt*  *Lt*1

*t*



*Lt*

*t*

.

* + - 1. The Gas-Puff Model.

This model (actually an entire circuit block, not simply a variable element) is similar to the cylindrical foil model, except that the user gives an initial outer and inner radius of a gas puff, as well as the length of the puff and its density. An initial mass is also specified, which can be the mass of a foil surrounding the puff at the outer radius. The user also gives the final pinch radius, which is usually specified as 1/10 of the outer radius. This model provides diagnostics of gas front velocity, kinetic energy and other electrical parameters.

The model calculates the inductance and time rate of change of the inductance assuming the current only flows in an infinitesimally thin shell, which is initially at the outer radius. As the shell collapses it picks up mass from the puff until the inner radius is reached. As the shell collapses further the mass remains constant. The collapse continues until the final pinch radius is reached.

* + - 1. The Dynamic Hohlraum Model

This model (actually an entire circuit block, not simply a variable element) is similar to the gas-puff model, except that two outer liners are added to the problem. The user specifies the radii and the masses of the two outer liners. The user specifies the outer radius of the foam load. The model provides for an optional third liner located at the outer radius of the inner foam target. An foam density must be specified. The model allows an inner radius to the foam, which must be greater than or equal to the final stagnation radius. The user defines the final stagnation radius, which is commonly specified as 1/10 of the outer radius. When the implosion reaches the final stagnation radius all dynamics cease. This model provides diagnostics of implosion velocity, implosion kinetic energy and other electrical parameters.

The model calculates the inductance and time rate of change of the inductance assuming the current only flows in an infinitesimally thin shell, which is initially at the outer radius. The inductance that is calculated is only the inductance between the starting radius of the outer liner and the radial location of the implosion at that time. All current is assumed to flow on the outside of the liner/s/. The model returns the inductance and the dL/dt for each time step. As the liners implode they accrete the other liners and finally accretes mass from the foam until the inner foam radius is reached. As the shell collapses further the imploding mass remains constant. The collapse continues until the defined minimum pinch radius is reached.

The model conserves monemtum for shell accretion and foam accretion assuming an inelastic collision. This effectively creates a drag term that must be accounted for in the acceleration routine. Energy that is lost in the accretion phases is included in the energy balance of the problem (variable inductance). In the real world, liner on liner stagnations can be seen from the energy radiated in the collision.

* + - 1. The Spherical Foil Model.

The spherical foil model (actually an entire circuit block, not simply a variable element) calculates the inductance and time-rate of change of the inductance for a portion of a spherical shell that collapses radially. It provides diagnostics of foil velocity, kinetic energy and other electrical parameters. The user specifies the initial radius, the angle of the sphere, the foil mass, and the final radius, which is usually set to 1/10 of the initial radius. The angle is specified in degrees, and must be between 0°and 180°. A value of 0° is meaningless, and an input of 180° will lead to an infinite inductance. It is assumed that the foil mass is loaded to achieve a simultaneous implosion at all angles. That is achieved by loading the foil mass as *1/cos θ*, where *θ* is measured from the central plane. Note that here *θ* is 1/2 of the model’s angle parameter.

The model calculates the magnetic acceleration, *a*, of the foil as a function of the current, *I*, both at time *t*, as

*at* =

*t*

*0 I2*

*2m*

where ** is the input angle (in radians), and *m* is the total foil mass. The foil mass is expressed as

*a*  *2*

*m* = 

–  *2*

 *cos**d*

where * =  0 /cos θ* and * 0* has units of mass per unit angle. The inductance, *L,* at time *t* is

*Lt* =

*2**0**rinit* – *rt* *ln*

*tan*-**-** + -**-** 

 *4 4* .

The time rate of change of the inductance, the value of *R* at time *t*, is again

 ---*L*-**-**

 *t*  *t*

*L* – *L*

= ----------------**-**

*t t-1*

Δ*t* .

* + - 1. The Dense-Plasma Focus (DPF) Model.

This is a model (actually an entire circuit block, not simply a variable element) of a Mather- type DPF. It is similar to the gas puff model and the standard z-pinch model, except that there are two different accretion modes: axial and radial. The axial “run down” phase is where a sheath propagates “mostly” axially. Once the sheath reaches the end of the anode then a radial implosion is possible. This radial implosion starts with a fraction of the mass in the axial sheath and then snowplows to the axis, accreting mass as it goes.

The user specifies the system parameters cathode radius, the anode radius, the anode length, and the gas-fill mass density. These are measured parameters and are generally not variable. The user sepecifies additional parameters: the fraction of the axial sheath used in the radial implosion, the minimum radius for the implosion, the initial mass of the axial sheath, and the sheath angle (from a plane perpendicular to the axis of the DPF. This model provides diagnostics of the radial velocity, radial kinetic energy and other electrical parameters.

The model calculates the inductance and time rate of change of the inductance per time step assuming the current only flows in an infinitesimally thin shell, which is initially at the input end to the DPF. As the shell collapses it picks up mass from the puff until the inner radius is reached. As the shell collapses further the mass remains constant. The collapse continues until the final pinch radius is reached.

The user specifies the sheath angle. This is done to match the observed sheaths seen in many Mather-type DPFs. It is NOT possible to match the measured current waveform without a sheath angle. In this version of the model, the sheath angle is purely linear. The angle of the sheath also generates velocity shear along the sheath and mass is lost during the implosion due to this shear.

Once the physical parameters of the driver and the load are placed in Screamer there are only two real ways to adjust the shape of the current waveform:

* + - * 1. Change the sheath angle to match the peak current, time to peak current, and general current waveshape, and,
        2. Change the fraction of the axial sheath mass with which the radial implosion starts.

The axial inductance is calculated with a standard co-axial inductance formula. In the case of the angled sheath, the sheath volume is divided into two portions: the upper part of the sheath that has an angle and the lower part for which the outer radius of the sheath has reached the cathode.

The total accreted mass is calculated in exactly the same way with the sheath volume divided into the upper, angled, portion and the lower, uniform, portion.

* + - 1. Tom Martin's Lossy Switch Model.

The model calculates resistive loss for air, water, SF6, oil, helium and hydrogen spark gap switches. The model has been carefully compared to detailed spark gap experiments and agrees very well with them over a large range of operating conditions. Using this model it is possible to construct detailed pulsed power circuits without first knowing or estimating switch losses.

This model uses the SCREAMER RLSeries block and modifies the series resistor. The model was developed by Tom Martin and is consistent with detailed switch energy loss experiments. It is described in detail in the paper "Energy Losses in Switches," by T. H. Martin, J. F. Seamen,

D. O. Jobe, in the Proceedings of the 9th Int. Pulsed Power Conference, 1993, p. 463. The model assumes a Braginskii formulation (referenced in Martin's paper) that describes the evolution of a spark channel in a dense gas or liquid. In keeping with Braginskii, the radius of

the channel is determined from the time-integral of the 2/3 power of the switch current *I*, multiplied by a constant that is a function of the mass density *0*, electrical conductivity **, and the ratio of the specific heats (embedded in the factor *K.*)

*a2* 

*t*

*1 / 3*

*K*



*2 / 3*

= *-------***-**

 

*0*

*I dt*

*0*

The resistance of the spark-gap channel is:

*Rchannel*

*Lgap*

= ----------**-***2*

*a* ,

where *Lgap* is the gap length. In a series of experiments ** appears to be constant and nearly the same for all breakdown media. Multiple switch channels and parallel switches are treated by dividing the resistance of one switch by the 1/3 power of the product of the number of switches and the number of channels

*Rn* =

*R1*

*----1--/-***-***3*

*n*

, where *n = nswitches*

*nchannels*.

The implicit assumption is that each switch channel carries an equal portion of the total current and that each channel has the same radius as a function of time.

* + - 1. The *Zflow* Plasma Opening Switch (POS) Model.

This model allows the user to specify a *Zflow* and switching time or threshold switching current for a resistor to ground, and thereby force the downstream current to be consistent with the *Zflow* definition. This model can be used for both plasma-opening switch calculations, and for determining plasma current losses in magnetically insulated transmission lines. The *Zflow* model forces a conductive loss to ground at the insertion point of the model. The user can also specify the time it takes for the switch to open to *Zflow.* Additional model parameters provide for clamping minimum and maximum conductance values. The model for the switch is shown

schematically in **Fig. 9**. It consists of a variable conductor to ground, the conductance of which is a function of the upstream current, the switch voltage, and *Zflow.*

*Iupstream*



*gswitch*

*I*

*switch*

*Vswitch*

*Idownstream*

Fig. 10. The ***Zflow*** POS model

Use of a *Zflow* parameter is a convenient way to handle plasma loss currents in a transmission line or a plasma opening switch. For a SCREAMER circuit *Zflow* is defined as

*Zflow* =



*I2*

*upstream downstream*

– *I2*

*Vswitch*

The model forces the downstream current to be consistent with a given *Zflow* by constructing a conductance to ground *gswitch* that satisfies the *Zflow* definition,

*gswitch* =

*Iswitch* =

*Vswitch*

*Iupstream* – *Idownstream*

*Vswitch*

By solving for *Idownstream* in the definition for *Zflow* and replacing *Idownstream* in this relation it is found that the switch conductance can be expressed as

*gswitch* =

*I*

*upstream* – *upstream*

*I*

*2*

– -------------**-**

 *Vswitch* *2*



*Zflow*



*Vswitch*

The code then uses the specified value of *Zflow* and the internal values of *Vswitch* and *Iupstream* to determine a conductance that forces a consistent downstream current. However, in actual practice, the value of *Vswitch* is very dependent on the local conductance to ground, and the model tends to oscillate and often not converge to the correct current. Since the currents don't vary as quickly as the voltage, a more stable method is to use the *Idownstream* from the previous time-step and remove *Vswitch* from the calculation. Thus, *gswitch* at time *t* can be expressed as

*gswitch,t* =



*Zflow upstream,t downstream,t-1*

*I2*

– *I2*

*Iupstream,t* – *Idownstream,t-1*

The model is checked by recalculating *Zflow* using the computed switch voltage, and upstream and downstream currents, for each time *t*. The recalculated value is available as a diagnostic.

* + - 1. The *Zflow* Loss Model.

This model is similar to the *Zflow* POS model, but has been specialized to allow a current loss consistent with MITL circuit model algorithms. This model forces a current loss at the insertion point described by Cliff Mendel’s *Zflow* definition. However, that loss is only turned on when Child-Langmuir emission in the MITL model is turned off. The model uses the magnetic insulation criterion to turn on *Zflow* loss, which is the criterion used to turn off Child-Langmuir emission in the MITL model. Magnetic insulation is achieved when

*c2 B2 E2*

 *1* +

*2*

*eV*  *mc2*

.

In both the MITL and the *Zflow* loss models the turn-on (or turn-off) is not abrupt, but rather follows a double exponential. One further restriction is that the electric field must also exceed the threshold for field emission. The field emission threshold is achieved by requiring that the field strength continuously exceed 300 kV/cm for a period of 5 ns.

With the *Zflow* loss model the user specifies the *Zflow* and the insertion point of the loss. The code then adjusts the resistance of a resistor to ground at the insertion point to produce a downstream current consistent with the specified *Zflow* and time-varying voltage at that point. The conductance is calculated as

*g* =



*Iupstream* – *Idownstream*

*Zflow upstream downstream*

*I2*

– *I2*

.

This model simulates loss of all of the plasma current in a transmission line at the insertion point. Therefore, it should generally only be used once in the line, although the coding will allow for multiple insertions. It would also probably be wise to place the insertion point at the location in the simulation where the loss is expected, such as the post-hole-convolute. If the loss is placed at a much larger or smaller radius, the turn-on criterion will be affected since that depends on the magnetic field strength. For example, if you expected most of the plasma current to be lost at a post-hole-convolute, but placed the *Zflow* loss near the load, the model would overestimate the loss in the leading edge of the pulse. At the post-hole- convolute Child- Langmuir emission would not be turned off as rapidly as *Zflow* loss would be turned on at the load.

* + - 1. The Resistive Wall Loss Model.

The resistive wall loss model, RWALL, estimates the resistive loss in the wall of current- carrying conductors at very high current densities. It is based on a model by Knoepfl, in the book*, Pulsed High Magnetic Fields1*, but modified by Bill Stygar2. This is only an estimate at this time since it does not use the actual current profile to determine the magnetic diffusion into the conductor, but rather assumes a linearly-rising current. Also, this routine does not consider current that is carried in a plasma sheath, nor does it consider effects due to melting, vaporization, or ionization of the conductor. The model assumes a constant-resistivity conductor. However, the resistance is scaled by the constant *C* that the user can specify to adjust for other materials or temperatures. The resistance *R* for a disk transmission line is described as

*R* = *C*

ln + ln 

*i i* ,

**0**0

** 3*t*

*router i*

*rinner i*

*lcylinder i*

*rcylinder i*





where ** is 7.2X10-7 Ohm-m for stainless steel*, t* is the time from the start of the current, *r* and *router* are the inner and outer radii of the disk conductors, and *lcylinder* and *rcylinder* are the lengths and radii of each individual coaxial conductor. All units are MKS. Because this formula is only valid for high current densities, the outer radius need only extend to where the peak current exceeds 1 MA/cm. For Z and ZX the outer radius is typically chosen as the radius of the post- hole convolute.

*0 inner*

The RWALL model, as it is presently written, assumes stainless-steel conductors. Therefore, the resistance calculated by the model is:

*R* = *C k* ln + ln 

*router i*

*rinner i*

*lcylinder i*

*rcylinder i*

*t*

*i*

*i*

,

where *k* = 1.708X10-7. This resistance is inserted in the circuit as a series resistance. The model allows for two disk conductors, and two cylindrical conductors.

* + - 1. The Resistive Wall Loss Model 2.

The new resistive wall loss model, R2WALL, estimates the resistive loss in the wall of current- carrying conductors at very high current densities. It is based on calculations published by Bill Stygar et al. in the Phys. Rev. ST Accel Beams **11**, 120401 (2008). This new model is an improvement from the simple RWALL model as the current waveform is now arbitrary. The details of the model are still only for stainless steel electrodes.

The model is implemented as follows:

1 Heinz Knoepfel, *Pulsed High Magnetic Fields*, (1970, North-Holland Publishing, Amsterdam).

2 W. S. Stygar, Bulletin of the American Physical Society, Division of Plasma Physics,

Conference held in New Orleans, LA, November, 1998.

+ ln

2**

*CHI*1

*t*

.5

*lcylinder i*

*rcylinder i*

*r disk outer i*

*r disk inner i*

+

1

*r*

2

*disk outer i*





*i i*

*R* = 

2

.25



2*CHI*2 *t I*

*i*

+ 1 

2 *i*

*lcylinder i*

*r*3

*cylinder i cylinder i*

1

*r*

2

*disk inner i*



,

where CHI = 3.38X10-8 and CHI = 3.7548X10-22. These constants are calculated from the fitting parameters in the paper. For our implementation we only have two cylindrical portions of the transmission line (inner and outer portion of a coax) and two disk sections of the transmission line (top and bottom disks). The individual specifications of the coax and disk sections are arbirary. One may have different coax lengths for the inner and outer pieces of the coax and one can have different disk start and stop radii for the two disks.

1 2

* + - 1. Classical Skin-Depth Model.

This model calculates the resistance of a conductor based on the cold skin depth. The calculation is done in the time domain and there is no heating in the model. Clearly, this model is not self consistent as a thin skin depth leads to local heating and a lower value of conductivity and more rapid diffusion of magnetic field and current into the conductor. We solve the magnetic diffusion equation as described by Sommerville et al.3 The problem and the solution are shown below:

2

1

 *Hz* =

*Hz*

*t*

*H x*, *t* = *I*

2*r*0

1 - erf *x*

2



*t*

Our approach is to solve for *H* as a function of depth time-step by time-step. In each time step we calculate the average value of *H* (essentially the integral divided by the depth) and determine the depth at which *H* is near the average *H*. The depth is then defined as the skin depth. The resistance is calculated using the material cold conductivity, the geometry of the conductor (coaxial or disk), and the skin depth. The model passes the new value of resistance back to the calculation.

There are several simplifying assumptions made. First, we are assuming 1-D cartesian diffusion. This assumption breaks down for conductor radii that are close the generated skin depth. In the case of a co-axial geometry, we calculate the resistance based on the dimensions of each conductor. In the case of the disk geometry, we assume the resistance of the whole disk to be 2X the resistance of a single disk. We assume that the transit time through the element is small compared with characteristic time scales in the problem. The present model is only valid

3 W. Sommerville, J. Gover, R. Sanchez, and J. Bou, Proceedings Electrical Insulation Conference and Electrical Manufacturing Expo, 2005, p. 383 (2005).

for an RLSeries circuit element. We have not validated this skin-depth model for the case of a lossy transmission line.

* + - 1. Wire Resistance Heating Model.

This model calculates the time resistance of a conductor based on the the resistance being a function of action. This is a simplified EOS based on Knopfel that does not include the correct EOS after vaporization. While simplified, this model demonstrates classical fusing, that is the rapid increase in voltage across a wire due to the rapid increase of wire resistance at melt.

The model requires detailed user EOS input. It is not limited to any single material but it only depends on the material EOS fitting the resistance-action table of the model. A simplified plot of the EOS is shown below in **Fig. 10**.

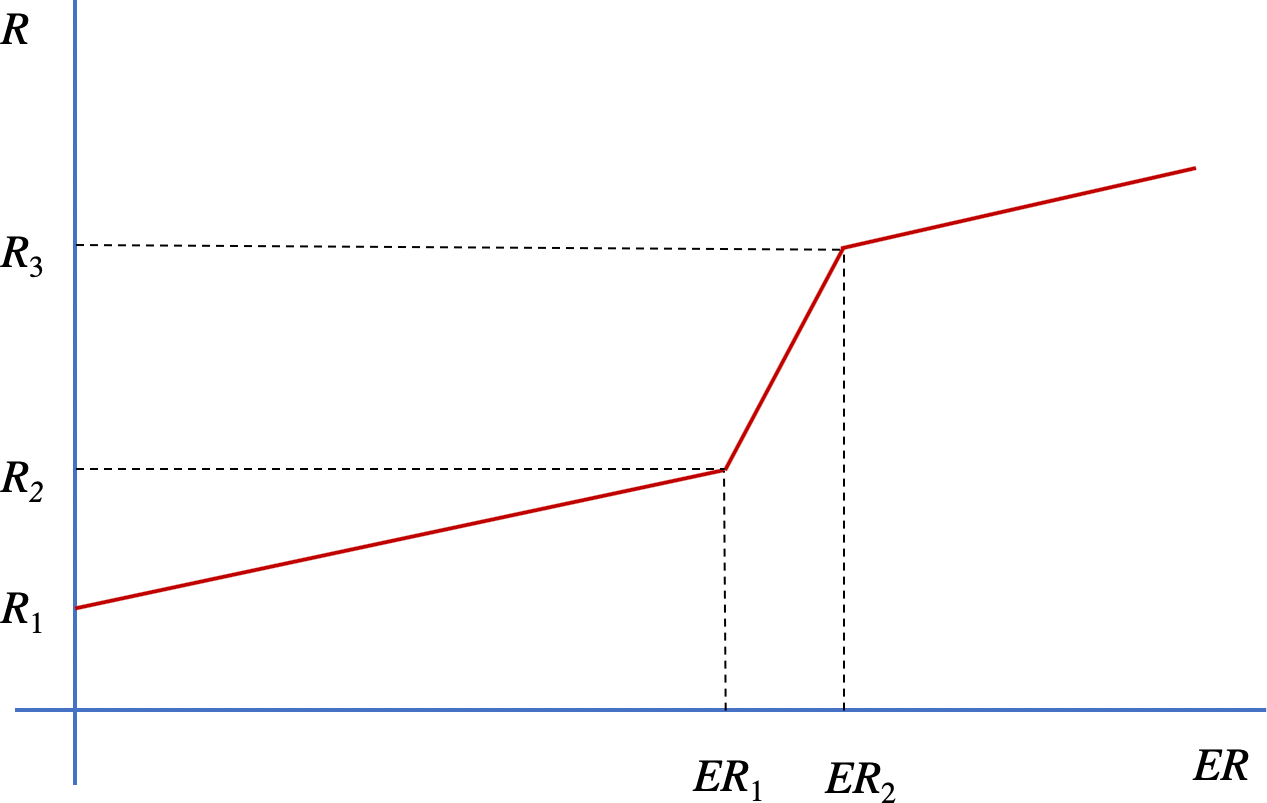
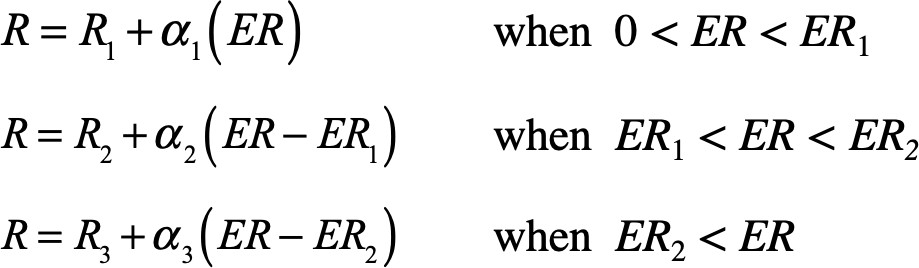


Fig. 11. A plot of the simplified wire EOS used in the Screamer model.

The user provides the input variables R1, ER1, ER2, and the three slopes, alpha1, alpha2, and alpha3. The portion of the curve between ER1 and ER2 is the region of melt. This curve is based on actual experimental data so it is self consistent with the change in resistivity with temperature, the change in the specific heat with temperature, and the slow expansion of the wire into a 1-bar environment.

The equations used for the three regions of the EOS are:



* + - 1. The Magnetic Flashover Inhibition (MFI) model.

The MFI (Magnetic Flashover Inhibition) model calculates the electric and magnetic fields across an insulator, and calculates an MFI flash-over criterion. When this criterion is exceeded, and when a critical electric field (50 kV/cm) is exceeded, the insulator is shorted. Specifically, the insulator will flash when the quantity

*Eparallel*

----------------**-**  *0.07* , or

*c B*

*Estack*

-----------**-**  *0.09* for a 45° insulator.

*cB*

Once the insulator flashes, it will stay shorted for the rest of the run. The routine also checks to see if the electric field is high enough to flash before the test is applied. Once the insulator flashes, the resistance to ground is identical to the exponential decay model. There is a parameter available so the routine can be run and the electromagnetic fields calculated without flashing the insulator.

* + - 1. The Radiation Yield Model.

This model (actually a diagnostic, not a variable element) only works with the the cylindrical foil model. It calculates K-line radiation yields using both Mosher-Krisnan-Qi and Whitney- Giuliani models. It is configured to give K-line yields for aluminum, argon, copper, krypton, and xenon.

* + - 1. Table Model

The table model is used to describe *R* or *G* values by allowing you to describe resistance as a function of time. You will be required to supply a time vs. resistance table on which a linear interpolation between points is performed.

In addition, the table model can be used to describe inductance *L* as a function of time only for the RLseries element (L2).

* + - 1. Multiple Collapsing Shell Model (NSHELL)

The NSHELL model calculates implosion velocity, radius, mass, inductance, time-rate-of- change of the inductance, and kinetic energy for a load with up to ten concentric shells.

**Inductance of multiple shells**. When current is not trapped in inner shells, the model calculates an equivalent inductance as the parallel combination of the inductances of all the shells.

*Leq* =

*N 1*

-*1*-**-**



*Lj*

*j* = *1*

where

*L* = *0l ln*

*2*

*rwall*

*rj*

Here *N* is the number of shells, *r* is the radius of the *jth* shell, and *l* is the length of the shell. Since the *r* all vary in time, the individual and equivalent inductances also vary in time. When the current becomes trapped, the equivalent inductance becomes just the inductance of first shell.

*j*

**Current division between the shells**. Before current is trapped, the currents in each shell are calculated by considering currents through parallel inductors. Thus, the current *i* in the *j*th shell is

*ij* =

*Leq*

*itotal* -----**-**

*Lj*

**Trapped Currents**. Currents on the inner shells become trapped when the time reaches the trapped-current time that is specified by the user. At that time the currents on the inner shells are frozen, and the total current flows only on the outer shell. The equivalent inductance is changed to the inductance of the outer shell. Since the voltage across the inductors is the sum of two terms, *Ldi/dt* and *idL/dt*, there can be a significant voltage spike when the inductance changes. This effect may be the explanation for the early-time spikes in the Compton-diode signals that are often observed on Z with multiple-shell loads.

**Conserved magnetic flux with trapped currents**. By requiring that the trapped flux on the inner shells be conserved as the shells collapse, it is possible to calculate the current on the *jth* shell, *j* = 2 and greater.

*i* *t*

= *i* *T*

*ln rj* – *1* *T* 

*rj**T* 

*j j* ---------------------------**-**

*rj* – *1**t*

*ln*

*rj**t* ,

where *T* is the trapped-current time, and *r(t)* is the radius of the *jth* shell at time *t*. It is clear from this relation that as the shells approach collision, the trapped currents become very large.

*j*

**Shell collisions**. Trapped currents can approach infinity as the shells collapse. Furthermore, the shells are not infinitesmally thin, as implicity assumed in this approach. Therefore, it makes sense to consider two shells to have collided when their radii are within a pre-defined distance. This distance has been arbitrarily defined in the NSHELL model as 50 m. When two shells approach to within this separation, the current on the inner shell is set to zero, its mass is added to the outer shell, and its radius is set as the outer shell radius.

By adding the masses of the two shells, an inelastic, snowplow-like collision is assumed. The model solves the force equation,

*B2 dv dm*

– --------*2* *rl* = *m* ----- + *v*-----**-**

*2**0*

*dt dt*

During the shell collapse, the mass derivative is zero. But at the time of collision, assuming the shells stick together, the mass derivative is infinite. By taking the limit at the time of the collision, it can be shown that the derivative can be accounted for by requiring momentum conservation during the collision. Therefore, *dm/dt* is always zero. But, at the time of collision the mass is incremented and the velocity is reduced. For a collision with the *jth* shell, this corresponds to a reduction in the velocity by

 *mi*

*v* *tj* +  = *v**tj* – *i*---=----*1*---*j*---–---*1*-----**-**

 *mi*

where *t* is the time of the collision with the *jth* shell, and ** is a small number. Similarly, the kinetic energy and the time rate of change of the inductance are also reduced by the same factor. The change in *dL/dt* shows up as a lower effective resistance.

*j*

Kinetic energy is not conserved in the collision. Kinetic energy can only be conserved if the shells bounce off each other, as in the collision of billiard balls. If that is the case, then the assumption of a snowplow-like collapse is not valid.

**Magnetic pressure drives the implosion**. As described in the previous paragraphs, the magnetic pressure provides the driving force on the shells. For each shell, there will be an inward pressure from the current on the outside of the shell, and an outward-directed pressure on the inside of the shell. The pressure on the *jth* shell is

*P* = *0 2 2*

*j* --------*2*---**-***2* *ij* – *ij* + *1*

*8* *r*

The acceleration of the *jth* shell is thus,

*a* = *0 2 2*

*j* --------------**-** *ij* – *ij* + *1*

*4**mjrj*

These expressions are exactly correct once currents are trapped, but only approximate early in time before trapping.

* + - 1. Electron-Beam Diode Model

SCREAMER incorporates an electron-beam diode model that has three options: a planar diode, a planar diode with edge emission, and a ring diode with finite area. We have used the following publications to generate the models.

C. D. Child, Phys. Rev. **32**, 492 (1911)

I. Langmuir & K. T. Compton, Rev. Mod. Phys. **3**, 191 (1931)

S. B. Swanekamp, et al., Phys. Plasmas **7**, 1514 (2000)

R. K. Parker et al., J. Appl. Phys. **45**, 2463 (1974)

A. Roy et al., Phys. Plasmas **16**, 053103 (2009)

The non-relativistic planar diode uses a simple 1-D treatment first described by Child and then Langmuir. Swanekamp restated the one-dimensional Child-Langmuir (1-D C-L) limited current in terms of the voltage normalized to the rest mass of the electron. Swanekamp provided a relativistic approximation for the 1-D planar case based on PIC simulations. We use a modification of the Langumir cylindrical diode formulation (Parker and Roy) to generate a model for ring diodes. We have adopted Swanekamp’s formulation to generate a relativistic ring diode model.

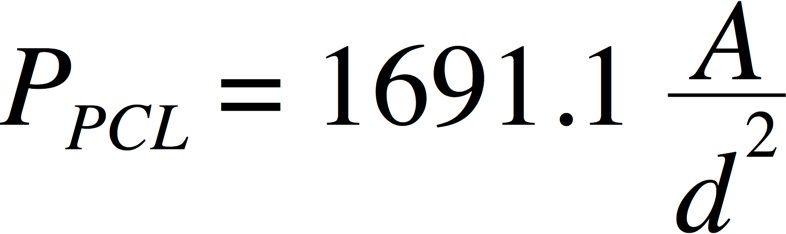
All of the model increase the current by 2x due to neutralization by protons. This factor of 2 is an average of the electron increase from the low energy increase to the relativistic incease. We designate this by the use of a subscript *p*.

The models actually pass the conductance *g* back to SCREAMER. The models provide the current limits and then, when dividied by the voltage (half time step) provide the conductance. Note: a planar case with edge effects is simply the sum of the planar model with an edge at the outer radius of the cathode. The ring diode case is simply the planar case with the ring area and two edges (inner and outer).

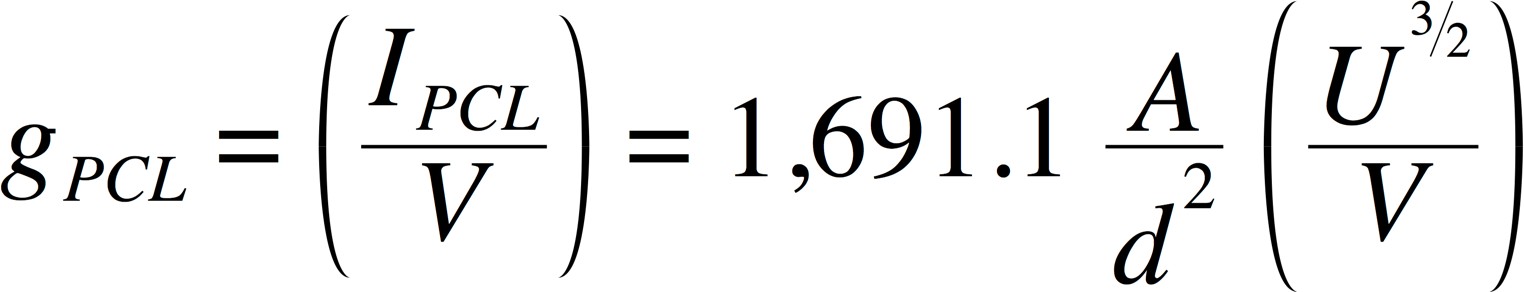
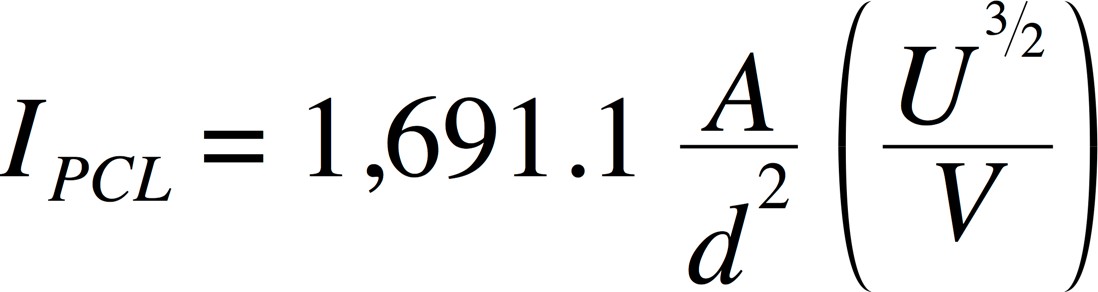
The model also includes gap closure. The gap closure depends on the inputted closure velocity. The cathode gap starts closing immediately and the anode gap starts closing 10 ns later.

Planar Model

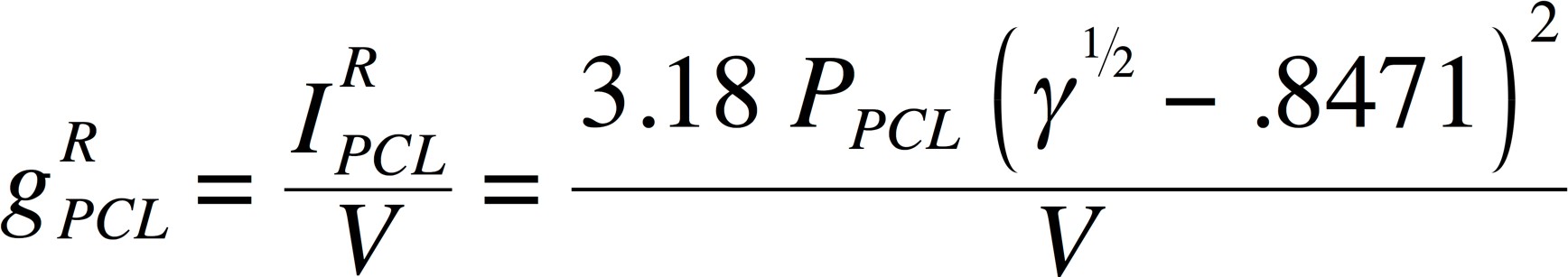
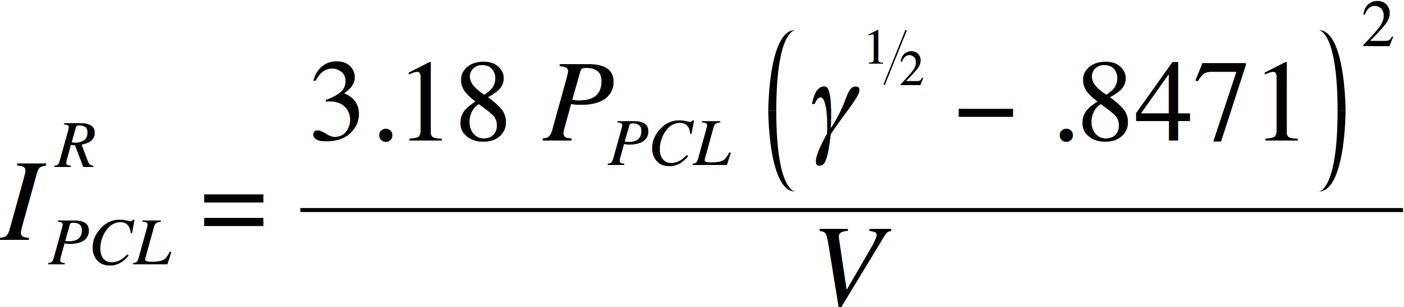
Non-Relativistic



Where *P* is the classical 1-D Perveance.

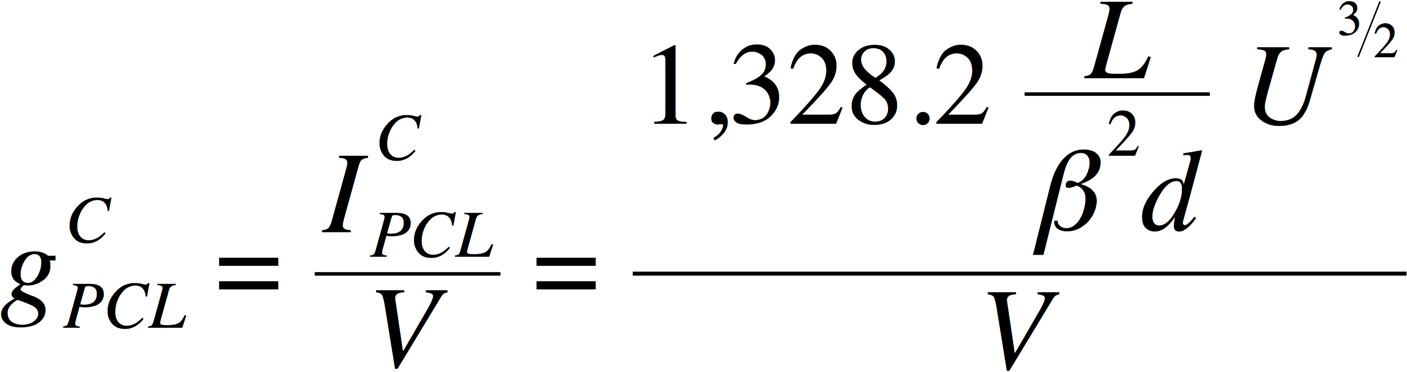
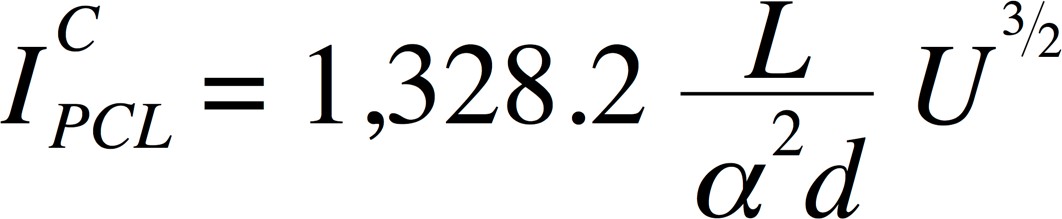


Relativistic

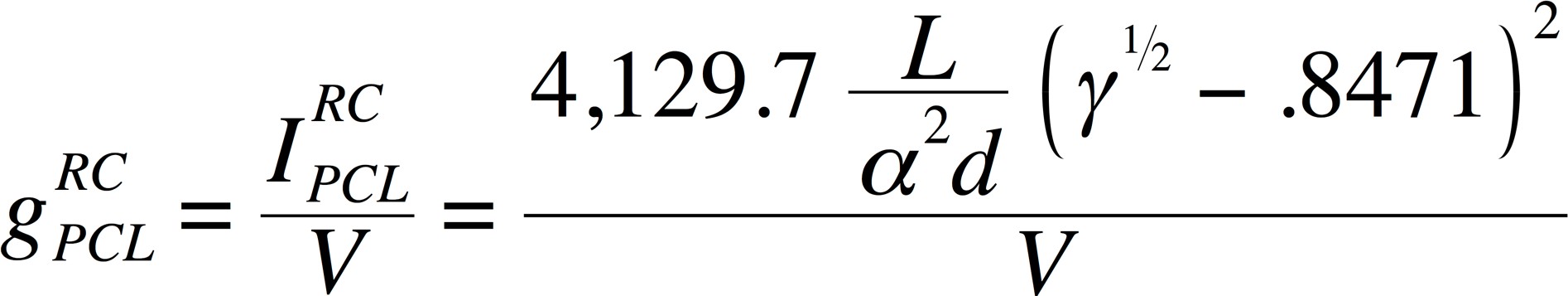
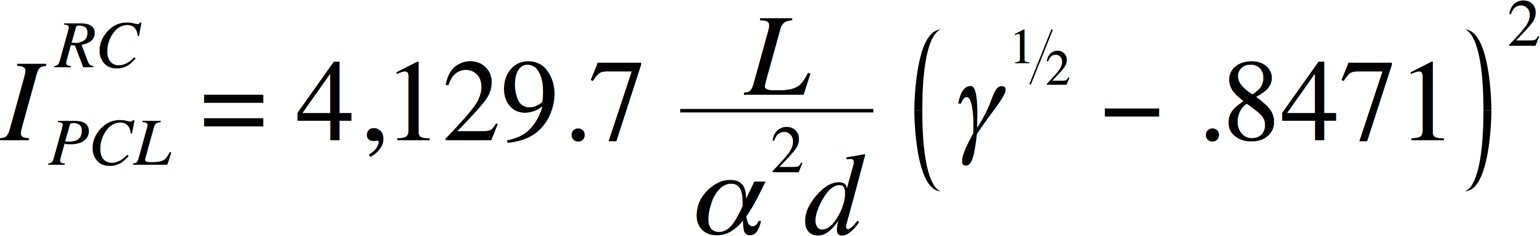


Edge Model

Non-Relativistic



Relativistic



### Switched Variable Element Models Contained in the SCREAMER “Library”

Currently, the SCREAMER model "library" contains the following switched variable element models. Additional documentation about these models is in **Section 5.3**.

* + - 1. Switched Variable Exponential Switch Model

The exponential switch model (gas-switch model) is used to describe the behavior of a resistor using the following equations. In this case a tabular value of the switch time *ti* is available from the Switch\_time setup parameter. Here *R(t)* is the resistance at time *t*:

*R( t ) = R o p e n* , when ( *t < t i* )

*Zswitche*–

*R**t* =

*Rclose* +

* = ( t – t i )/ *

*1* – *e*– + *Z*

*switch*

*10*–*6*

, when (*t > ti*)

*R o p e n* , *R c l o s e* , *t i* , **  and *Z s w i t c h* ar e all us er sp ecif i e d .

* + - 1. Switched Variable Decay Switch Model

The decay switch model is used to describe a closing switch. The resistance remains constant at *Ropen* until the time specified in the Switch\_time setup parameter. This is designated by *ti*. At that time, the value of the resistance decays exponentially to *Rclose* as described by the following equations, where *R(t)* is the resistance at time *t*:

*R( t ) = R o p e n* , when ( *t < t i* )

*R( t ) = R + ( R – R )e- * , when ( *t > t* )

*c l o s e o p e n c l o s e i*

* = ( t – t i )/ *

*Ropen*, *Rclose*, *ti*, and ** are all user specified.

* + - 1. Switched Variable Rise Switch Model

The rise switch model is used to describe an opening switch. The resistance remains constant at *Rclose* until the time specified by *ti*. The switch time *ti* is determined by the Switch\_time setup parameter. At that time, the value of the resistance rises exponentially to *Ropen* as described by the following equations, where *R(t)* is the resistance at time *t*:

*R( t ) = R c l o s e* , when ( *t < t i* )

*R( t ) = R c l o s e + ( Ro p e n – Rc l o s e )(1 - e - * *)* , when *(t > t i )*

* = ( t – t i )/ *

*Ropen*, *Rclose*, *ti*, and ** are all user specified.

1. **Input/Output Overview of SCREAMER**
   1. **Input to SCREAMER**

The minimum input required by SCREAMER includes the description of the circuit which is denoted as the " SCREAMER input file". Optionally, you may also include as input the names of one or more user-written subroutines describing variable circuit elements.

### Run-Time Options

When you execute SCREAMER, you will be prompted to enter the name of your SCREAMER input file. With the UNIX version, you may also enter on the command line the names of user- written subroutines. SCREAMER will automatically compile the subroutines and link and run a new version of SCREAMER. Again, you will be prompted to enter the name of your SCREAMER input file

* + 1. **The SCREAMER Input File**

Before you execute SCREAMER, you must have already created a SCREAMER input file.

**Chapter 5** provides complete details on how to construct a SCREAMER input file.

### User Subroutines

SCREAMER provides you the flexibility of writing your own Fortran subroutines to describe a variable circuit elements which are not included in SCREAMER library. See **Chapter 6** for instructions on writing user subroutines.

* 1. **Output from SCREAMER**

SCREAMER provides (1) a SCREAMER log file which contains useful summary information concerning your SCREAMER job, and (2) selected circuit output values as a function of time (diagnostics) written to a variety of file formats for post-analysis.

* + 1. **The SCREAMER Log File**

Every time you run a SCREAMER job, a log file will be created which includes (1) a listing of the circuit defined in your input file, (2) an energy check at specified time intervals, and (3) run-time statistics. See **Section 8.1** for a detailed discussion of the SCREAMER log file.

The Log File is invaluable when looking for errors in the run deck or in individual elements.

### Circuit Parameter Output

There are many file formats you may request for examining a circuit parameter’s behavior in time. See **Section 7.2** for more information.

* + - * *FILE*

Creates a separate ASCII (text) file for each diagnostic call in the run deck.

* + - * *TABLE*

Creates a separate ASCII file for each diagnostic, but puts it in the form of a SCREAMER compatible table which may be pasted into another SCREAMER input file (a voltage versus time table for a voltage source specification, for example).

* + - * *TXT*

Creates a single ASCII file containing all diagnostics that may be analyzed using any graphics package, as well as some spreadsheet packages. The data format is “space separated variable” between diagnostic columns.

* + - * *PFF*

Creates a single Portable File Format (PFF) file containing all the diagnostics. Data in PFF files is stored in a compact, binary form in such a manner that it can be read without translation on a wide variety of computers. This is not available for the IBM-PC or MacIntosh versions. (Not presently supported.)

* + - * *CSV*

Creates a single "comma separated variable" file containing all the diagnostics. This file format can be used directly with many spreadsheet programs. This file format is an ASCII format that organizes the data into rows and columns, which can be read into programs such as EXCEL or Kaleidagraph (Macintosh and Windows).

* + - * *SFC*

Creates a single "Standard de Ficheirs Communs" (Common Standard for Files) file containing all the diagnostics. This file format is an ASCII format that organizes the data into rows and columns, which can be read into such programs as *EXCEL* or *XMGR*.

**Figure 11** summarizes the input/output processing flow for SCREAMER. The solid arrows indicate required input and automatic output. The dotted arrows indicate optional input/ output.

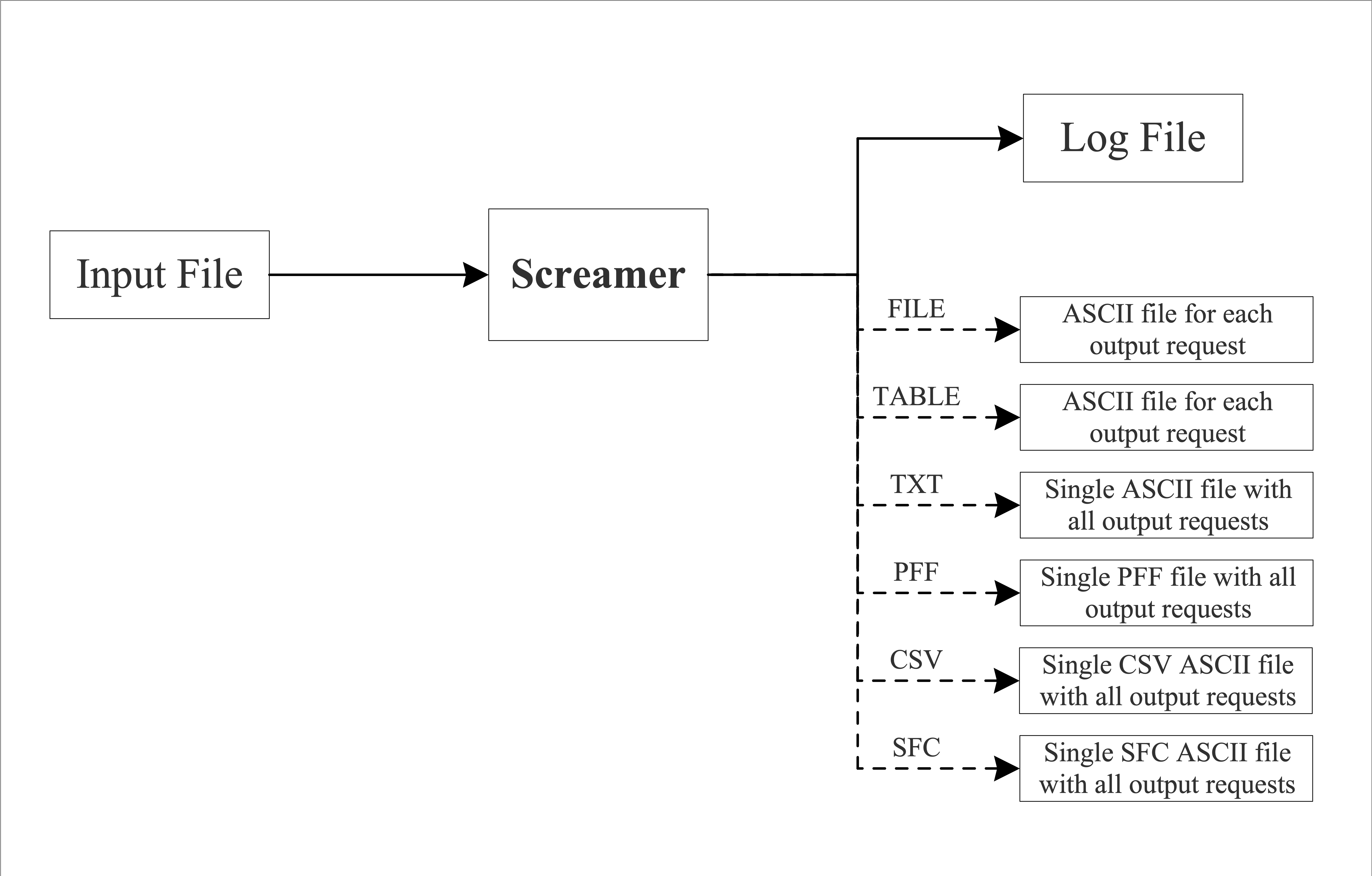


Fig. 12. Input/Output Flow for SCREAMER

# Constructing an Input File

Once you have defined a SCREAMER problem, you are ready to translate that problem into a SCREAMER input file. The SCREAMER input file consists of keywords and numerical values.

## Conventions

Some general guidelines for building a SCREAMER input file are:

* Most keywords may be abbreviated to 3 letters. The allowed abbreviations will be un- derlined in the text that follows. In fact, SCREAMER only reads the first three letters so that anything typed after these in a keyword is ignored. You might want to use this feature for in-line comments.
* All numerical values may be entered in free format. Further, when parsing input, SCREAMER is case-insensitive.
* Use MKS units.
* Some numerical values are optional and these will be shown in parentheses in the input format sections. If you do not enter an optional parameter, it will be assigned a default value.
* Keywords and numerical values may be separated by blanks (the recommended method), by a comma and blanks or by TAB characters.
* Each line of the data file is generally one complete segment of information, and must not be separated into two or more lines. Conversely, you may not combine two or more lines into one.
* If the first nonblank character on a line is the ’!’ character, the entire line is regarded as a comment line and will be ignored.
* If the first nonblank character on a line just after an output request is the ’$’ character, the rest of the line will be used as a title identifying the output request. If this line is not after an output request line, it is taken to be a comment line.

## Setup Conditions

Before a SCREAMER problem can begin execution, you need to set a few initial parameters in your input file. These are the iteration time step, the total simulation time, the default transmission line resolution time, and the number of times to print out the energy balance of the circuit. (The status of the circuit at *t = 0* is always given.) Also, you can specify whether all or just one of the time iteration cycles should be performed. This is useful in checking an initial run of a complicated setup before performing the full simulation. In addition, you may

specify the maximum number of points to be stored for each *FILE*, *TXT*, *PFF, CSV* or *SFC*

output request.

The first line of the input file is the title. There are no keywords on this line. Whatever you type on the line is taken to be the title of your problem. The title will appear on any output SCREAMER generates. The next 6 lines specify the parameters which control the solution algorithm, status reports, and diagnostic frequency. The first five are mandatory and the next is optional.

1. The iteration time step Tim*e-step t*

*t* is the value for the iteration time step. Note that the run time for any file will be

roughly inversely proportional to the time step chosen. In many cases a time step that is surprisingly large will work well.

1. The default resolution time for transmission lines

*Resolution-time tres*

*tres is* the default resolution time used for breaking lines into smaller segments. This parameter can be overridden for a particular line by specifying a resolution time for that line. The number of segments each line will be divided into is /2 *tres*. The run time for any problem will scale inversely as the resolution time.

1. The simulation end time

*End-time tend*

*tend* is the value for the time to end the simulation

1. The number times to print the circuit status

*Number-prints np*

*np* is the number of times to list the circuit status in the SCREAMER log file during the simulation and *np* > *1*. Note that you will always get a circuit status at *t = 0*, regardless of *np*.

1. The number of iteration cycles (time steps) to execute

*Execute-cycles ec*

*ec* is the keyword *One* or *All*. This specifies whether to execute only the first time iteration cycle in the simulation and then quit or to execute all of the cycles. If only one cycle is to be executed, the end-time entry will have no effect and a print-out of the circuit status will be done at *t = 0* and after the first iteration.

1. The maximum points to store for *FILE, TXT, PFF, CSV* and *SFC* output requests.

*Max-points Mp*

*Mp* is the maximum number of points to store for a *FILE, TXT, PFF, CSV* or *SFC* output request. If this line is not included, *Mp* will be set to the maximum allowed value as listed in **Chapter 10** or as set for your local installation. *TABLE* output requests have a different limit, also listed in **Chapter 10**, but it cannot be modified.

1. Switch table data values that can be used in the exponential and decay (and rise) resistor models.

*Switch-times t1*

*t2*

*.*

*.*

*.*

*ti*

*Last-Entry*

A list (vector) of switch times can be provided in SCREAMER. This *optional* call places the switch times in a common block that is available to all of the elements that are capable of using these data entries. At this time the only elements that can access the list of switch times are the Decay, Rise, and Exponential Resistor Models. If the call is NOT used in the setup conditions the switch times content of the common block is set to 0.00.

## Circuit Description

### Circuit Branch Locations

SCREAMER incorporates *branches* to increase circuit complexity because the basic SCREAMER topology only allows RCGround’s, RLSeries’s, PISection, and TRLines in series. For example, using an End Branch that contains a single RLS circuit block (and an RCG that is set to 0.0) effectly allows SCREAMER to use a resistor/inductor to ground. Similarly, a Top Branch that contains only an RCG is effectively allowing a resistor/capacitor parallel combination in series. This branch structure allows aribrarily complex circuits.

Before SCREAMER V4.0 only Level 1 (L1) and Level 2 (L2) branches were allowed. The L1 branch is the “Main” Branch or the first branch that is called. Other Branches can be called from within the L1 branch. Branches that exit from the L1 branch are DEFINED to be L2 branches. As of SCREAMER V4.0 branches that exit from L2 branches are allowed. These are DEFINED to be L3 branches. Branches are read in and defined in exactly the order that they are called in the L1 and L2 branch. For example, An L1 “main” branch that calls three (L2) branches defines the order that the L2 branch definitions must have immediately following the end of the L1 branch.

There are three input line types which are used to specify the locations of the circuit branches:

1. The branch line call is Bra*nch*

This line specifies that a new branch is to be input and that the lines that follow contain information about the circuit blocks are related to that separate branch. The Branch content ends when the next branch command is given or the deck ends.

1. The first branch type is the Top Branch. The top branch call is Top*branch*

This line specifies that a top branch will exit the *previous* block at the *last two* nodes of that block.

1. The second branch type is the End Branch. The end branch call is End*branch*

This line specifies that an end branch will exit the *previous* block at the *last* node of that block. Note that an end branch may not be attached to the last block of the main branch.

Branches will be numbered in the *exact* order they appear in the SCREAMER input file. For example, in the Demon model (**Section 5.7.2**), the second top-branch line in the first branch, the Level 1 main branch, indicates that the third branch in the input file is a top branch which exits the -section block in branch 1 (this is the block that is listed previous to the second top- branch line).

An example of SCREAMER branch structure follows in a sample Run Deck. (We do not show the setup paramaters for clarity and omit block details for simplicity.) We use a single Top Branch and a single End Branch just to show how they are called.

!

! Branch #1 Level 1 Main Branch

!

Branch

{Block 1}

!

! Branch #2 called

!

Topbranch

{Block 2}

{Block 3}

!

! Branch #3 called

!

Endbranch

{Block 4}

!

! END Branch #1 Level 1 Main Branch

!

! Branch #2 Level 2 Top Branch

!

Branch

{Block 1}

!

! END Branch #2 Level 2 Top Branch

!

! Branch #3 Level 2 Endbranch Branch

{Block 1}

!

! END Branch #3 Level 2 End Branch

!

The example reduces the Run Deck contents to the minimum for structural clarity. The branches called from the “main” branch have their contents defined in separate branches that follow in *exactly the order* that they are called in the “main” branch. In our example above, each of the L2 branches has a single block. This is the simplest possible branch. A real branch will have many circuit blocks, up to the limit of the code. It is *very* helpful to label the branches as they are called AND as they are defined AND as they end to be able to keep everything straight.

### Circuit Blocks

There are currently 16 input line types which specify the blocks in the circuit branches. (See

### Figs. 3 through 7.)

1. Lossless transmission line

*TRLine Zvar * *Zin (Zout) (tres)*

*Zvar* is the keyword *Linear* or *Exponential* and specifies whether the impedance of the line varies linearly or exponentially. ** is value for the length of the transmission line, *Zin* and *Zout* are the input and output impedances. If *Zout* is not specified, then SCREAMER will set *Zout* = *Zin*, giving the line a constant impedance. *tres* is the resolution time for this line, and if not specified, the default resolution time specified by your Resolution-Time entry is used.

1. Lossy transmission line

*LOSsyline * *Zin R1 R2 (Zout) (tres)*

** is value for the length of the transmission line, *Zin* and *Zout* are the input and output impedances. *R1* is the total shunt resistance of the length of transmission line specified by **. *R2* is the total series resistance of the length of transmission line specified by **. If *Zout* is not specified, then SCREAMER will set *Zout* = *Zin*, giving the line a constant impedance. *tres* is the resolution time for this line, and if not specified, the default resolution time specified by your Resolution-Time entry is used.

1. Racetrack MITL

*MITL Cir Gap * Z (*tres) (eturnon)*

*Cir* is the circumference of the MITL feed and *Gap* is the MITL gap. ** is the line delay and *Z* is line impedance (assumed constant over the length of the line). Again, *tres* is the line’s resolution time and if not entered, the default resolution time specified by your Resolution-Time entry will be used. *eturnon* is the electric field, in units of volts per meter, at which cathode emission is turned on. If not specified, it is set to 200 kV/cm (2X107 volts/m)

1. Perveance MITL

*PMITL Per * Z (*tres)*

*Per* is the perveance of the MITL, ** is the time delay, and *Z* is the impedance of the line (assumed to be constant over the length of the line). *tres* is the line’s resolution time and if not entered, the default resolution specified by your Resolution-Time entry time will be used.

1. The -section

*PISection R1 C1 R2 L2 R3 C3*

*R1*, *C1*, *R2*, *L2*, *R3*, *C3* are the values of the resistances, capacitances, and inductance for the -section block shown in **Figure 6**.

1. The resistor and capacitor to ground

*RCGround R1 (C1*)

*R1* and *C1* are the values of the resistance and capacitance shown in **Figure 6**. *C1 = 0* if not entered.

1. The series resistor and inductor

*RLSeries R2* (*L2*)

*R2* and *L2* are the values of the resistance and inductance shown in **Figure 6**. *L2 = 0* if not entered.

1. The adder block

*ADDer*

This block has no parameters besides the keyword and is useful when adding the voltage from a secondary branch to that of the main branch. In this case, a "topbranch" line would directly follow the "adder" line. The adder block is equivalent to an RLSeries block with L2 = 0.0 H and R2 = 1.0X106 Ω

Four source block types are available for putting energy into the circuit. (Please refer to **Figure 7**.)

1. The beginning-of-main-branch voltage source

*Voltsource function R2 L2*

1. The end-of-branch voltage source

*Vendsource function R2 L2*

1. The beginning-of-main-branch current source

*Currsource function R3 C3*

1. The end-of-branch current source

*Cendsource function R1 C1*

In the source block input lines, *function* is the keyword *SSQ* (a sine-squared voltage or current waveform), *SIN* (a sine waveform), *LSF* (waveform described by a least squares polynomial), or *TAB* (a waveform described by a table of values multiplied by a scale factor).

1. If the ’*SSQ*’ current or voltage waveform is selected, then one line follows with the form (where *F* is the value of the voltage or current):

*SF tpulse (tdelay*)

*SF* is the scale factor for multiplying the sin-squared function*, tpulse* is the duration of the waveform (half-period of the sin function), and *tdelay* is the time at which to begin entering the waveform into the circuit (a time delay on the source).

*F = 0, if t < tdelay* or *t > tdelay + tpulse.*

Otherwise:

*F* = *SF*

*t* – *tdelay*  *2*

*sin* -------------------------**-**

*tpulse* .

*tdelay = 0* if not entered.

1. If ’*SIN*’ is selected, then one line follows with the form:

*SF tpulse (tdelay*)

*SF* is the scale factor for multiplying the sin function, *tpulse* is the duration of the waveform (full period of the sin function), and *tdelay* is the time at which to begin entering the waveform into the circuit (a time delay on the source).

*F = 0, if t < tdelay* or *t > tdelay + tpulse.*

Otherwise:

*F* = *SF*

*t* – *tdelay**2*

*sin* -----------------------------**-**

*tpulse* .

*tdelay = 0* if not entered.

1. If ’*LSF*’ is selected, then one line follows with the form:

*A0* (*A1*) (*A2*) (*A3*) (*A4*) (*A5*) (*A6*) (*A7*) (*A8*) (*A9*)

*F* = *Aiti*

*i*

*Ai = 0* if not entered.

1. If ’*TAB*’ is selected, then from at least 4 lines follow. The first line has the form:

*SF tdelay*

*SF* is the scale factor by which all interpolated voltages or currents will be multiplied and *tdelay* is the source time delay. The rest of the lines are of the form:

*ti, Vi*

These specify the time vs. unscaled voltage or current table on which a linear interpolation is performed. You must specify between 2 pairs up to the maximum allowed. **Chapter 10** lists this value (maximum number of points for a *TABLE* request) which may be overridden for your local installation. These lines are followed by the line which specifies the end of the table:

*Last-entry*

Note that you may "paste" a time vs. voltage table or time vs. current table which was created during a previous SCREAMER run using the *TABLE* output option.

### Initial Conditions

There are five input line types which are used to specify initial voltages or currents in the circuit. Only one initial condition may be specified per block. In these lines, *voltage* is the value of the initial voltage and *current* is the value of the initial currrent. See Figures and for the definitions of *C1*, *C3*, and *L2* for the various circuit blocks.

1. Initial voltage on *C1 Initial VC1 voltage*
2. Initial voltage on *C3 Initial VC3 voltage*
3. Initial voltage on a lossless transmission line

*Initial VTRL voltage*

1. Initial current in a lossless transmission line

*Initial ITRL current*

1. Initial current in *L2 Initial IL2 current*

### Variable Elements

There are many variable element models in the SCREAMER library. In general, two lines are used to specify variable elements and they must appear immediately after the block in which the element is located. The table model requires at least 3 lines. NOTE: Only one variable element is allowed per block. See **Section 3.3.1** for a detailed description of the variable element models and their parameters.

1. The exponential switch model *VARiable element* EXP*-model Ropen Rclose tswitch * *Zswitch*

*element* is the keyword *R1*, *R2*, or *R3* and specifies which element is variable. The second line gives the various parameters which describe the behavior of the resistor.

1. The switched variable exponential switch model

*SVAriable element* Exp*-model Ropen Rclose ti * *Zswitch*

*element* is the keyword *R1, R2*, or *R3* and specifies which element is variable. The second line gives the various parameters which describe the behavior of the resistor. *Ropen* is the value of resistance prior to the switch opening. *Rclose* is the final value of the resistance after the swtich opens. *ti* is the time that the switch closes that is obtained from the Switch\_time time in the setup parameters. ** is the time constant of the decay.

1. The decay switch model *VARiable element* DEC*ay-model Ropen Rclose tswitch *

*element* is the keyword *R1, R2*, or *R3* and specifies which element is variable. The second line gives the various parameters which describe the behavior of the resistor.

1. The switched variable decay switch model

*SVAriable element* DEC*ay-model Ropen Rclose ti *

*element* is the keyword *R1, R2*, or *R3* and specifies which element is variable. The second

line gives the various parameters which describe the behavior of the resistor. *Ropen* is the value of resistance prior to the switch opening. *Rclose* is the final value of the resistance after the switch opens. *ti* is the time that the switch closes that is obtained from the Switch\_time time in the setup parameters. ** is the time constant of the decay.

1. The rise switch model (resistor) *VARiable element* RIS*e-model Ropen Rclose tswitch *

*element* is the keyword *R1, R2*, or *R3* and specifies the variable element. The second line gives the various parameters which describe the behavior of the resistor.

1. The switched variable rise switch model (resistor)

*SVAriable element* Ris*e-model Ropen Rclose ti *

*element* is the keyword *R1, R2*, or *R3* and specifies which element is variable. The second

line gives the various parameters which describe the behavior of the resistor. *Ropen* is the value of resistance prior to the switch opening. *Rclose* is the final value of the resistance after the switch opens. *ti* is the time that the switch closes that is obtained from the Switch\_time time in the setup parameters. ** is the time constant of the rise.

1. The magnetic switch model of a saturable core inductor

*Variable element* MSW*itch-model*

*PF Ri Ro* W *H1 Hsat Hrev Bsat*

*element* is the keyword *L2* and specifies the inductor as the variable element. The second line gives the parameters which describe the behavior of the inductor.

1. The plasma opening switch models

*Variable element* PS1*-model*

*tswitch K Rmax Rmin*

Or:

*Variable element PS2-model*

*Qswitch K Rmax Rmin*

*element* is the keyword *R1* or *R3* and specifies the variable element. The second line gives the parameters which describe the behavior of the resistor. The first model uses switching based on time and the second uses switching based on accumulated charge.

1. The classic diode model

*Variable element DIOde-model*

*V1 I1 V2 I2 V3 I3 V4 I4 V5 I5 V6 I6*

*element* is the keyword *R1* or *R3* and the second line gives the parameters describing the measured diode curve. The pairs of points describe the diodes V-I curve with the assumption that a point exists at V = 0 and I = 0.

1. The Slutz diode model

*Variable element SDIode-model*

*td Rmax Rmin A G v Gmin (Pmratio)*

*element* is the keyword *R1* or *R3* and the second line gives the parameters describing the diode’s behavior. *Pmratio* is an optional parameter. If you do not include this parameter, it will be set to 1.

1. The Cylindrical Foil Model Block

*CYLFOIL rmax length mass rmin*

*rmax* is the initial foil radius, *length* is the foil length, *mass* is the foil mass, and *rmin* is the final radius. All units are MKS. Typically, *rmin* is chosen to be 1/10 of *rmax*. Usually this model is the last element in the simulation. However, since it is a series R and L block it must be followed with a block to complete the circuit, such as:

RCGround 1.0e-12 0.0

1. The Gas-Puff Model Block

*GASPUFF rinitial length density rfinal rinner initialshellmass*

*rinitial* and *rinner* are the initial outer and inner radius of the gas-puff, *length* and *density* are the gas-puff length and density, *rfinal* is the final pinch outer radius, and *initialshellmass* is the mass of a foil surrounding the puff at the outer radius*.* The final pinch radius is usually specified as 1/10 of the initial outer radius. Again, since this model is a series R and L block and is usually the last element in the simulation, it must be followed with a block to complete the circuit.

1. The Dense-Plasma Focus (DPF) Model Block

*DPFmodel router rinner anode-length density rmass\_frac rmin initialshellmass theta*

*router* and *rinner* are the initial outer and inner radii of the cathode and anode, repectively, *anode-length* is the length of the anode stalk, *density* is the static gas fill, *rmass\_fraction* is the fraction of the axial sheath mass that is available to the radial implosion, *rmin* is the minimum pinch radius, *initialshellmass* is the mass of a foil surrounding the puff at the outer radius, *theta* is the angle of the sheath relative to a plane perpendicular to the DPF axis*.* The final pinch radius *rmin* is usually specified as 1/10 of the initial outer radius. The fraction of the axial sheath mass assigned to the radial implosion is small, usually less than 1X10-4. The sheath angle *theta* is usually greather than 45°. Again, since this model is a series R and L block and is usually the last element in the simulation, it must be followed with a block to complete the circuit.

1. The Dynamic Hohlraum Model Block

*DYHOHLRAUM length rfoil1 mfoil1 rfoil2 mfoil2 router mfoil3 density rinner rmin*

*length* is the dynamic hohlraum length, *rfoil1* is the initial radius of the outer liner, *mfoil1* is the mass of the outermost liner, *rfoil2* is the initial radius of the second liner, *mfoil2* is the mass of the second liner, *router* is the radius of the third liner AND the outer radius of the dynamic hohlraum foam, *mfoil3* is the mass of the third liner that is located on the surface of the foam, *density* is the density of the dynamic hohlraum foam, *rinner* is the inner diameter of the dynamic hohlraum foam, *rmin* is the final stagnation radius and is usually specified as 1/10 of the initial outer radius. Again, since this model is a series R and L block and is usually the last element in the simulation, it must be followed with an RCG block to complete the circuit.

Caveats: All parameters MUST be entered. Keep in mind the logic of the radii: *rfoil1*

> *rfoil2* > *router* > *rinner* ≥ *rmin*. The mass of a liner can be set to zero except the outer liner. Note, there is no reason to run a problem with this model if one is going to set the outer liner to zero mass.

1. The Spherical Foil Model Block

*SPHFOIL rinit angle mass rfinal*

*rinit* is the initial spherical foil radius, *angle* is the angle of the spherical foil shell, *mass* is the initial foil mass and *rfinal* is the final implosion radius. The units of *rinit, mass*, and *rfinal* are MKS and the units of *angle* are degrees. Again, since this model is a series R and L block and is usually the last element in the simulation, it must be followed with a block to complete the circuit.

1. Tom Martin's Lossy Switch Model

*RLS initial\_resistance inductance*

Var*iable* R2 SWI*tch\_model*

*Dielectric switchtime gaplength pressure nswitches nchannels*

The model can be used any number of times in a input file. *dielectric* is an ASCII input variable that can be one of the following:

*H2O*, *OIL*, *SF6*, *AIR*, *HE*, or *H2*

Note also that oil is the only dielectric in this list that has not been correlated with data. However the usefulness of the dielectric and the fact that its density lies between water and the gasses means that the model should make accurate predictions. *switchtime* is the time the switch opens, *gaplength* is the length of the gap, *pressure* is the spark gap pressure in units of atmospheres. For the liquids, the pressure should be set to 1 atmosphere. *nswitches* is the number of parallel switches and *nchannels* is the number of parallel channels in each switch.

1. The *Zflow* Plasma Opening Switch (POS) Model

*RCG initial\_resistance capacitance*

*Variable R1* POS*\_model*

*tswitch currentsw topen zflow gmin gmax crowbarflag*

*tswitch* is the switch time, *currentsw* is the current threshold for switching, *topen* is the time it takes the switch to open, *zflow* is the *Zflow* at the time the switch opens, *gmin* and *gmax* are the minimum and maximum conductance of the switch, and *crowbarflag* is a flag to specify whether the switch should be shorted (crowbarred) if the voltage

reverses. If *crowbarflag* is set to one, the switch will crowbar on voltage reversal. For the switch to open both *tswitch* and *currentsw* must be exceeded. This feature allows switching at either a specific time or at a specific current. To switch at a specific current, set *tswitch* to a small value, and *currentsw* to the desired current. Conversely, if *currentsw* is set to a small value the switch will open at *tswitch*.

This model can also be used to calculate plasma current loss in an magnetically insulated transmission line. The model is used for this purpose by setting *tswitch* and *topen* to zero, and setting *zflow* to an appropriate value.

*crowbarflag* is used only for POS switch applications. This flag should be set for POS switches, except when you wish to see how much energy could be passed through the switch if the voltage could be prevented from reversing. If the voltage does not reverse, this flag has no effect.

1. The *Zflow* Loss Model

*RCG initial\_resistance capacitance*

*Variable R1* Zlo*ss-model*

*zflow gap radius gmin gmax N*

*zflow* is the value of *Zflow*, *gap* is the transmission line gap, *radius* is the insertion point radius, *gmin* and *gmax* are the minimum and maximum conductances, and *N* is the number of parallel lines. *gap* is used to calculate the electric field, and *radius* and *N* are used to calculate the magnetic field. This model simulates loss of all of the plasma current in a transmission line at the insertion point. Therefore, it should generally only be used once in the line, although the coding will allow for multiple insertions.

1. The Resistive Wall Loss Model

*RLS* 0.0 0.0

*Variable R2 RWAll\_model*

*tstart disk1inner disk1outer disk2innter disk2outer radius\_cyl\_1 length\_cyl\_1 radius\_cyl\_2 length\_cyl\_2*

The model requires a current start time (*tstart*), which assumes a linearly-rising current. Since the start time is often not known before the run, it is necessary to run the circuit first without the model to determine the start time, and then to put that time in for subsequent runs. The model also requires an inner and outer radius of two disk conductors(*disk1inner*, *disk1outer*, *disk2inner*, and *disk2outer*), and lengths and radii of two cylindrical conductors (*radius\_cyl\_1*, *length\_cyl\_1*, *radius\_cyl\_2*, and *length\_cyl\_2*). The outer radius of the disk should not extend past the point where the peak current density is below 1 MA/cm. The second input, the constant (*constant*) is a

scale factor that can be used to adjust the resistance for temperature effects, for example. The constant should be set to 1.0 for room temperature stainless steel, but can be set for higher values if much higher temperature is expected. Remember, however, that wall resistance scales as the square-root of the resistivity in this model

1. The Modified Resistive Wall Loss Model

*RLS* 0.0 0.0

*Variable R2 R2W*a*ll\_model*

*tstart cyl\_length\_inner cyl\_length\_outer cyl\_r\_inner cyl\_r\_outer disk\_upper\_r\_inner disk\_upper\_r\_outer disk\_lower\_r\_inner disk\_lower\_r\_outer*

The model optionally can define a current start time (*tstart*), before which there is no wall resistance. Setting tstart to 0.0 is usually the best approach. The model requires the lengths, the inner radius, and the outer radius of the cylindrical sections of a transmission line. If either of the cylindrical lengths are set to 0.0 then that cylindrical portion of the model is removed. The second line of inputs are two disk conductors (upper and lower) where we need to define the inner and outer radii of each disk. Setting *disk\_upper\_r\_inner = disk\_upper\_r\_outer* will effectively remove the effect of the upper disk transmission line. The same can be done to the lower transmission line.

1. The Skin Depth Resistive Wall Loss Model

*RLS* 0.0 0.0

*Variable R2 RSKin\_model*

*Sigma depth cyl\_length cyl\_r\_outer cyl\_r\_inner disk\_length disk\_outer\_radius*

The model requires the material cold conductivity, the material depth allowed in the problem (1 mm is typical), lengths, the inner radius, and the outer radius of the cylindrical sections of a transmission line. If the cylindrical length is set to 0.0 then that cylindrical portion of the model is removed. The second line of inputs are two disk conductors where we need to define the outer radii of the disk. Setting *disk\_length* to

0.0 will effectively remove the disk element from the calculation. It is assumed that the user will use one of the other of the geometries.

1. The Wire Resistance Heating Model

*RCG initial\_resistance capacitance*

*VARiable R1 RCOnd\_model*

*Initial\_R, Action\_ER1, Action\_ER2 Alpha1, Alpha2, Alpha3*

The model can be used for RLSeries (R2) and RCGround (R1) circuit elements.

1. The Magnetic Flashover Inhibition (MFI) Model

*RCG initial\_resistance capacitance*

*Variable R1 MFI\_model*

*radius gap gmin gmax N flag*

*radius* and *gap* are the radius and gap of the insulator, *gmin* and *gmax* are the minimum and maximum conductance, *N* is the number of parallel insulators, and *flag* specifies whether the insulator will crowbar when the flashover criterion is exceeded. Normally *flag* is set to one, but it can be set to zero for testing. This will allow calculating the electric and magnetic fields and flashover criterion without causing the insulator to flash.

1. The Radiation Yield Model

This model provides estimates of K-shell radiation yields from the cylindrical foil model. Only output requests are needed; there are no model parameters. See **Section**

**5.5** for a complete description of the format of these output requests.

1. The Table Model (resistor or inductor)

*VARiable element TABle-model*

where *element* is the keyword *R1*, *R2*, *R3*, or *L2* and specifies the variable element. At least 4 lines follow. The first line has the form:

*SF tdelay*

where *SF* is the scale factor by which all resistance (inductance) values will be multiplied, and *tdelay* is the resistance delay. The following lines are of the form:

*ti Ri* (or *Li*)

The (*ti*, *Ri*) pairs make up the "time vs. unscaled resistance" table. Similarly (*ti*, *Li*) pairs make up the "time vs. unscaled inductance" table. A minimum of 2 pairs must be specified. The last pair is followed by the line which specifies the end of the table:

Las*t-entry*

Note that you may "paste" a time vs. resistance (inductance) table which was created during a previous SCREAMER run using the *TABLE* output option. (See **Section 5.4**.)

1. The Multiple Collapsing Shell (NSHELL) Model

*NSHELLMODEL length fradius ak\_gap trap\_time*

*r1 m1*

*r2 m2*

.

.

.

*rn mn*

LAS*T ENTRY*

where *length* is the load length, *fradius* is the final radius of the implosion, *ak\_gap* is the radial gap between the outer shell or wire array and the return-current wall, and *trap\_time* is the trap time. The pairs of data are the radius and mass of each shell. IMPORTANT: Do not separately include the initial inductance of the load. The model caclulates the inital inductance. Up to ten pairs of radius and mass can be input. Since NSHELLMODEL is a series R and L block it must be followed with a block to complete the circuit, such as:

RCG 1.0e-12 0.0

1. The Electron-Beam Diode Model

*RCG initial\_resistance capacitance*

*VARiable R1 EDIode\_model*

*Diode\_type gap enhancement velocity radius\_outer (radius\_inner)*

Where *diode\_type* (1, 2 or 3) is either a planar diode (1), a planar diode + edge (2), or a ring diode (3). The *gap* is the anode/cathode spacing. The *enhancement* is the electric field enhancement on the cathode. The *velocity* is the plasma expansion velocity from both the anode and the cathode. The *radius\_outer* is the radius of the planar diode or the outer radius of the ring diode. The *radius\_inner* is the inner radius of the ring diode. We need the inner radius in order to calculate the area of the ring diode.

## Output Requests

*FILE*, *TABLE*, and *PFF* requests have the following basic format: Format: *Type Parm (time\_flag) (tstart*) (*tstop*)

*Type* is one of six keywords referring to the type of output: *FILE*, *TABLE*, *TXT*, *PFF, CSV* or *SFC*. *time\_flag* is one of the two keywords: *WHOLE* or *HALF*. This

flag determines whether the data for the requested output will be reported on the whole time step or the half time step (average of previous time step and current time step). *time\_flag* is only valid for *FILE*, *TABLE*, and *PFF* output requests. If you do not specify *time\_flag* for these three output request types, SCREAMER will default to *HALF* for *FILE* and *TABLE* output requests and *WHOLE* for *PFF* output requests. *TXT* and *CSV* requests ignore *time\_flag* and always store data on the half time step. *SFC* requests ignore *time\_flag* and always stores data on the whole time step. Note that *time\_flag* may be omitted even if *tstart* and *tstop* are included on the line.

*tstart* and *tstop* specify a time window for examining the parameter and if not given, *tstart* is set to zero and *tstop* is set to *tend*. For *TXT*, *PFF, CSV,* and *SFC* requests, the *tstart* and *tstop* parameters are ignored; the time window is set to the simulation time window, *t = t0* to *t = tend*. Note that for non-*PFF* requests, if the time window begins at *t = t0* (a whole time step) and the output request is for values on the half time step, the first value that will appear in the output request is at *t = t0*. All others will be on half time steps.

*TXT, CSV*, and *FILE* requests have the following basic format: Format: Type Parm (SCALE #)

*Type* is one of six keywords referring to the type of output: *FILE*, *TABLE*, *TXT*, *PFF, CSV* or *SFC*. Note, the old text file keyword was UFO, while being supported for backward compatibility is will be deprecated in the future. *SCALE* is the keyword indicating that that output request will be scaled by the real number given. For example, if you which to have a current request be outputted in MA then the multiplier would be 1.0e-6.

The *Parm* keyword specifies the parameter that is to be examined. For any block (**See Figure 12.** ) *Parm* may be:

*VIN* or *VOUT* (the voltage across the input or output of the block)

*IIN* or *IOUT* (the current flowing into or out of the block) *PIN* or *POUT* (the power flowing into or out of the block) *EIN* or *EOUT* (energy flowing into or out of a block)

*QIN* or *QOUT* (charge flowing into or out of a block)



**VOUT**

**VIN**

**IOUT**

**IIN**

Fig. 13. Some Plotting Conventions for Circuit Blocks For source blocks only (see **Figure 13**), *Parm* may be:

VSR*C* (the source voltage)

ISR*C* (the source current)

PSR*C* (the source power)

ESR*C* (the source energy developed)

QSR*C* (the source charge developed)



|  |  |
| --- | --- |
| **ISRC**  **R2 L2**  **VSRC V**  **Beginning-of-Main-Branch Voltage Source** | **ISRC**  **R2 L2**  **V VSRC**  **End-of-Branch Voltage Source** |
| **ISRC I**  **R3 C3 VSRC**  **Beginning-of-Main-Branch Current Source** | **ISRC I**  **VSRC R1 C1**  **End-of-Branch Current Source** |

Fig. 14. Plotting Conventions for Circuit Source Blocks

For transmission line circuit blocks only (**Figures 3 through 4**), *Parm* may be: ELI*NE* (energy stored in the transmission line)

PLI*NE* (power stored in the transmission line) EDL*INE* (energy dissipated in the transmission line) PDL*INE* (power dissipated in the transmission line)

and for the "racetrack" MITL circuit block only, *Parm* may be:

ALO*SS* (loss current density to the anode due to Child-Lang- muir emission)

For the -section, the RC to ground, the RL series, and the source blocks, *Parm* may be: Vxx (the voltage across an element)

Ixx (the current through an element)

Pxx (the power dissipated or stored in an element)

Exx (the energy dissipated or stored in an element)

Qxx (the charge on or passing through an element)

xx (the value of the element)

where *xx* is *R1, C1, R2, L2, R3,* or *C3.* (See **Figures 3 through 6** for the block by block definitions of these element names.) One may also get output of the following for *L* and *C*:

FL2 (the magnetic flux in *L2*)

*L2EFF* (the value of

C1E*FF* (the value of

C3E*FF* (the value of

*L2 I*  *I* )

*C1V*  *V* )

*C3V*  *V* )

For any block in which a variable element is described by a user subroutine, *Parm* may be: Ux (the value of a user variable)

where *x* is an integer in the range 1 < *x* < 10. See **Section 6.4** for more information.

## Additional Output Requests for Models of Variable Elements

Many of the models which describe variable elements have specific, model-dependent diagnostics which can be extracted using output requests. Again, the previous conventions for

output requests are used. The user simply specifies the model-dependent *Parm* keyword. A list of models and their available output requests are shown below.

### Cylindrical and spherical foil models

FRA*D* (foil radius)

FVE*L* (foil velocity)

FAC*C* (foil acceleration)

FKE (foil kinetic energy)

### Gas-puff model, DPF, and Dynamic Hohlraum models

GRA*D* (shell radius)

GVE*L* (shell velocity)

GAC*C* (shell acceleration)

GKE (shell kinetic energy)

### Lossy switch model

FCH (channel radius)

* + 1. ***Zflow* POS model**

ZFL*OW* (calculated *Zflow*)

GZF*LOW* (switch conductance)

### MFI model

EFL*D* (electric field)

BFL*D* (magnetic field)

XMF*I* (flashover criterion)

### Radiation Yields for the Cylindrical Foil Model

YWL (Aluminum, Whitney formulation)

YWA (Argon, Whitney formulation)

YWC (Copper, Whitney formulation)

YWK (Krypton, Whitney formulation)

YWX (Xenon, Whitney formulation)

YML (Aluminum, Mosher formulation)

YMA (Argon, Mosher formulation)

YMC (Copper, Mosher formulation)

YMK (Krypton, Mosher formulation)

YMX (Xenon, Mosher formulation)

If no output requests are made, the values are not calculated, and the model runs faster.

* + 1. ***Zflow* Loss Model**

CZL*OSS* (Calculated *Zflow*)

GLO*SS* (Conductance to ground)

### Multiple Collapsing Shell (NSHELL) Model

Note that since mass is not constant, it is possible to output the accrued mass as a function of time.

SKE (Shell kinetic energy)

SVE*L* (Shell velocity)

SRA*D* (Shell radius)

SAC*C* (Shell acceleration)

SMA*SS* (Shell mass)

SR1 (Shell radius 1)

SR2 (Shell radius 2)

SR3 (Shell radius 3)

SR4 (Shell radius 4)

SR5 (Shell radius 5)

SC1 (Shell current 1)

SC2 (Shell current 2)

SC3 (Shell current 3)

SC4 (Shell current 4)

SC5 (Shell current 5)

## Comments, Labels, and Titles

Comment lines, title for output requests, and user variable labels may be entered in the SCREAMER input file.

1. Comment lines

Format: !*comment placed here*

Any line except the first line that has a ’!’ as the first nonblank character will be taken to be a comment line and will be ignored.

1. Titles for output requests Format: $*title placed here*

Any line that has a ’$’ as the first nonblank character and that occurs immediately after

an output request in the SCREAMER input file is taken to be a title for that output request. The allowed length for a title is 22 characters.

For a *FILE* request, the title will be used as the name of the variable.

For a *TABLE* request, the title is simply included in the output file as an extra comment. For a *TXT* output request, the title is used as the column label for each variable.

For a *PFF* output request, the title will be used as the PFF comment.

For a *CSV* output request, the title appears as the column label for each variable.

For a *SFC* output request, the title appears as a column heading within the *TITCOL* key word.

If the title line occurs after a line which is not an output request, it will be taken to be a comment line and will be ignored. If no title is entered for an output request, SCREAMER will create a its own title, identifying the block by its branch and block indices. An example of a standard TXT output request and title follows:

txt IOUT SCAle 1.0e-6

$MITL\_Current (MA)

Here we as for the current out of the prior element and scale the current by 10 . The label shows the name of the output and indicates the scaling.

-6

1. User Variable Labels

*Format: Ulabel first\_word second\_word*

The user variable label line is used to explicitly define the label to be used for output requests for user variables. For more information, see **Section 6.4**.

# Input File Summary

This chapter is a summary of the format of a SCREAMER input file. For a detailed discussion of the SCREAMER input file, see **Chapter 5**.

### Title Line (anything typed on line 1)

**Setup Condition Lines (Lines 2 through 7):**

*Time-step t Resolution-time tres End-time tend Number-prints np*

*Execute-cycles ec* {*ec* is *One* or *All*} *Max-points Mp* {optional}

### Branch Location Lines:

*Branch* {A branch starts here}

*Topbranch* {A top branch exits previous block}

*Endbranch* {An end branch exits previous block}

### Block Lines:

*Trline Zvar * *Zin* (*Zout*) (*tres)* { *Zvar* is *Lin* or *Exp*} *MITL Cir Gap * *Z* (*tres) (eturnon)*

*PMITL Per * Z (*tres) Pisection R1, C1, R2, L2, R3, C3 Rcground R1 (C1*)

*Rlseries R2* (*L2*) Add*er*

*Voltsource function R2 L2 Vendsource function R2 L2 Currsource function R3 C3 Cendsource function R1 C1*

If *function* is *SSQ* or *SIN*, one card follows with the form:

*SF tpulse ( tdelay* )

If *function* is *LSF*, then one card follows with the form:

*A0 ( A1* ) ( *A2* ) ( *A3* ) ( *A4* ) ( *A5* ) ( *A6* ) ( *A7* ) ( *A8* ) (

If *function* is *TAB*, then one card follows with the form:

*SF*

then at least 2 cards with the form:

*ti Vi*

followed by the card: Las*t-entry*

*A9* )

### Initial Condition Lines:

Ini*tial* VC1 *voltage* Ini*tial* VC3 *voltage* Ini*tial* VTR*L voltage* Ini*tial* IL2 *current* Ini*tial* ITR*L current*

### Exponential Switch Model Lines:

*Variable element Exp-model {element* is *R1*, *R2*, or *R3*}

*Ropen Rclose tswitch*  *Zswitch*

### Decay Switch Model Lines:

*Variable element Decay-model* {*element* is *R1*, *R2*, or *R3*}

*Ropen Rclose tswitch* 

### Rise Switch Model Lines:

*Variable element Rise-model* {*element* is *R1*, *R2*, or *R3*}

*Ropen Rclose tswitch* 

### Magnetic Switch Model Lines:

*Variable element Mswitch-model* {*element* is *L2*}

*PF Ri Ro* W

*H1 Hsat Hrev Bsat*

### Plasma Opening Switch (time-switch) Model Lines:

*Variable element PS1-model* {*element* is *R1* or *R3*}

*tswitch* K

*Rmax Rmin*

### Plasma Opening Switch (charge-switch) Model Lines:

*Variable element PS2-model* {*element* is *R1* or *R3*}

*Qswitch* K

*Rmax Rmin*

### Slutz Diode Model Lines:

*Variable element SDiode-model* {*element* is *R1* or *R3*}

*td Rmax Rmin* A G v *Gmin* ( *Pmratio* )

### Cylindrical Foil Model Block:

*CYLFOIL rmax length mass rmin*

### Gas-Puff Model Block:

*GASPUFF rinitial length density rfinal rinner initialshellmass*

### Dynamic Hohlraum Model Block:

*DYHOHLRAUM length rinit mfoil1 rfoil2 mfoil2 router mfoil3 density rinner rmin*

### Spherical Foil Model Block:

*SPHFOIL rinit angle mass rfinal*

### Lossy Switch Model (for RL series block):

Var*iable* R2 Swi*tch\_model*

*dielectric switchtime gaplength pressure nswitches nchannels*

{*dielectric* is *H2O*, *OIL*, *SF6*, *AIR*, *HE*, or *H2}*

### Zflow POS Model (for RC to ground block):

*Variable R1* POS*\_model*

*tswitch currentsw topen zflow gmin gmax crowbarflag*

### Zflow Loss Model (for RC to ground block):

*Variable R1* Zlo*ss-model zflow gap radius gmin gmax N*

### Resistive Wall Loss Model (for RL series block):

RLS *0.0 0.0*

*Variable R2 Rwall\_model*

*tstart contant disk1inner disk1outer disk2inner disk2outer radius\_cyl\_l length\_cyl\_l radius\_cyl\_2 length\_cyl\_2*

### Cold Skin Depth Model (for RL series block):

*RLS* 0.0 0.0

*Variable R2 Rskin\_model*

*Sigma depth cyl\_length cyl\_r\_outer cyl\_r\_inner disk\_length disk\_outer\_radius*

### Wire Resistance Heating Model (for RLseries and RCGround blocks):

*RLS* 0.0 0.0

*Variable R2 RCOnd R1 ER1 ER2*

*alpha1 alpha2 alpha3*

### Magnetic Flashover Inhibition Model (for RC to ground block):

*Variable R1 MFI\_model radius gap gmin gmax N flag*

### Table Model Line:

*Variable element* Tab*le-model* {*element* is *R1*, *R2*, *R3, or L2*} Then one card follows with the form:

*SF tdelay*

then at least 2 cards follow with the form:

*ti Ri*

followed by the card:

Las*t-entry*

### The Multiple Collapsing Shell (NSHELL) Model

NSH*ELLMODEL length fradius ak\_gap trap\_time*

Then one to 10 cards follow with radius/mass pairs:

*radius mass*

followed by the card: Las*t-entry*

### User Written Model Cards:

*Variable element User-model*, or

*Variable element Usn-model*

where *n* is an integer in the range

*1*  *n*  *4* .

### Output Request Cards:

*Type Parm (time\_flag) ( tstart* ) ( *tstop* )

*Type* is:

*FILE, TABLE, TXT, PFF, CSV*, *SFC*

*Parm* is:

*VIN, IIN, PIN, EIN, QIN,*

*VOUT, IOUT, POUT, EOUT, QOUT, VSRC, ISRC, PSRC, ESRC, QSRC,*

*ELINE, PLINE, EDLINE, PDLINE, ALOSS,*

*Vxx, Ixx, Pxx, Exx, Qxx, xx* {*xx* is *R1,R2,L2,R3,C3*} *L2EFF, FL2, C1EFF, C3EFF*,

*Ux* { *1*  *x*  *10* },

*FRAD, FVEL, FACC, FKE* {cyl., sph. foils},

*GRAD, GVEL, GACC, GKE* {gas-puff},

*FCH* {lossy switch},

*ZFLOW, GZFLOW* {Z-flow POS},

*EFLD, BFLD, XMFI* {MFI},

*YWL, YWA, YWC, YWK, YWX, YML, YMA, YMC, YMK, YMX* {radiation yields},

*CZLOSS, GLOSS* {Zflow loss}

*SKE,* SVE*L,* SRA*D,* SAC*C,* SMA*SS,* SR1*,* SR2*,* SR3*,* SR4*,* SR5*,* SC1*,* SC2*,* SC3*,*

SC4*,* SC5 {multiple collapsing shell}

*time\_flag* is:

*WHOLE* or *HALF*

### Comment Lines:

! *comment*

### Output Request Titles:

$*title*

### User Variable Labels:

Ula*bel first\_word second\_word*

# Example Input Files

This section contains some example SCREAMER problems and their corresponding SCREAMER input files. These examples are included with the SCREAMER distribution. See your system administrator for the exact location of these files on your computer.

### A Simple Capacitor Discharge Circuit

The listing of the SCREAMER input file which describes a simple capacitor discharge circuit is given below. The output requests will result in saving the history of 5 variables in a TXT file: the voltage on the initially charged capacitor VC1, the energy in the initially charged capacitor EC1, the current through the series resistor IR2, the energy in the series inductor EL2, and the power dissipated in the load resistor PR2:

Capacitor Discharge

! 2014-03-12 RBS

!

Time-step 0.25e-9 Resolution-time 2e-9 End-time 1e-6 Number-prints 5 Execute-cycles all Grids no

Echo-setup no Max-points 3001

!

!Start ciruit definition

! BRANCH

RCG 1e+12 560e-9

Initial VC1 80e3 TXT VC1

$Voltage(V) TXT EC1

$Ecap(J)

!

Rlseries 0.0 64e-9 TXT IR2

$I(A) TXT EL2

$E(J)

!

!Load to ground

!

Rcground 0.072 0.0 TXT PR1

$Power(W)

!

! End circuit

### A Dynamic Hohlraum Circuit

The listing of the SCREAMER input file which describes a dynamic hohlraum circuit is given below. We use a simple *sin* voltage waveform voltage source with a series resistor and inductor to model a pulsed-power driver and we use the hohlraum block as the dynamic load. The output requests will result in saving the history of 5 variables in a TXT file: the open circuit driver voltage VSRC, the liner radius GRAD, the liner acceleration GAC, the liner velocity GVEL, and the liner current IR1:

Dynamic Hohlraum Test

TIME-STEP 0.01E-9 RESOLUTION-TIME 0.5E-9 END-TIME 0.2E-6

NUMBER-PRINTS 1

EXECUTE-CYCLES ALL

ECHO NO

MAX-POINTS 2000

!

! Rick Spielman

! 2014-10-23

!

! Start with a simple sin driver, 0.2 Ohm, 10 nH , dynamic hohlraum

! BRANCH

!

VOLTSOURCE SIN 0.0 0.0

4.0e6 200e-9

!

TXT VSRC

$VOC\_in

!

! Water line impedance

!

RLS 0.20 0

!

! MITL inductance

!

RLS 0 10.0E-9

!

! Load inductance

!

RLS 0 2.0e-9

!

!DYHohl

!length rliner1 mliner1 rliner2 mliner2 router mliner3 density rinner rmin

!

DYH 2.0e-2 2.0e-2 1.0e-6 1.0e-2 1.0e-6 0.5e-2 1.0e-6 5.0e-2 2.0e-3 2.0e-3

!

TXT GRAD

$Rad(m) TXT GAC

$Acc(m/s2) TXT GVEL

$Vel(m/s)

TXT IR2

$Cur(A)

!

RCG 1E-12 0.

### The Marx Circuit

The Marx circuit is given by the circuit diagram in **Figure 14**. This is a simple circuit, consisting of one -section block with the first shunt capacitor charged to 5.1 MV initially.



16 nF

1.4 K

22 nF

5.1 MV

12 H

2.5

Fig. 15. Block Circuit Diagram of Marx Module

The listing of the SCREAMER input file which describes the Marx circuit is given below. The output requests will result in saving the history of 5 variables in a CSV file: the voltage on the initially charged capacitor, the voltage and current in the load resistor and the power and energy dissipated in the load resistor:

Marx model, 5ns time step, CSV output types only, no user models Time-step 5e-9

Resolution-time 5e-9

End-time 1000e-9

Number-prints 5

Execute-cycles all

Max-points 500

BRANCH

!Enter the pisection block and set the initial voltage on C1.

Pisection 1e+12 22e-9 2.5 12e-6 1400 16e-9 Initial VC1 5e+6

csv VC1 whole

$Source capacitor voltage csv VR3 whole

$Output voltage csv IR3 whole

$Output current csv PR3 whole

$Output power csv ER3 half

$Output energy

### The Demon Model

The Demon model consists of a main branch with four top branches, three of which consist solely of a shunt capacitance across a series inductor and variable resistor representing switch capacitance. The listing of the SCREAMER input file is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Demon Model With 1ns | Time Step, CSV | output types | only, | no user models |
| Time-step | 2.0E-9 |  |  |  |
| Resolution-time | 5.0E-9 |  |  |  |
| End-time | 1500.0E-9 |  |  |  |
| Number-prints | 1 |  |  |  |
| Execute-cycles | all |  |  |  |
| Max-points | 501 |  |  |  |
| BRANCH |  |  |  |  |
| Pisection 1e+12 | 22.3e-9 2.4 | 10.3e-6 | 1e+12 | 0 |
| Initial VC1 | 5.7e+6 |  |  |  |
| Pisection 775 | 0 0.3 | 2.56e-6 | 3e+3 | 0 |
| Variable R1 | Exp-Model |  |  |  |
| 775 | 22 5e-6 | 5e-9 | 20 |  |
| Trline Linear | 16.6e-9 7.41 |  |  |  |
| Trline Linear | 34e-9 4.6 |  |  |  |
| Rcground 3000 |  |  |  |  |
| Trline Linear | 17e-9 4.6 |  |  |  |
| Trline Linear | 14e-9 5.2 |  |  |  |
| Adder |  |  |  |  |

TOPBRANCH

Trline Linear 20e-9 3.89

Rcground 1e+3

Trline Linear 20e-9 3.89

Pisection 1e+4 1.06e-9 1e+6 100e-9 1e+4 0.0

Variable R2 Exp-Model

1e+6 0.1 1.108e-6 10e-9 6.05

CSV R2

$Resistance of second gas switch TOPBRANCH

|  |  |  |  |
| --- | --- | --- | --- |
| Trline  Rcground | Linear  1e+4 | 10e-9 | 2.16 |
| Trline | Linear | 10e-9 | 2.16 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Pisection  Variable | 1e+4  R2 | 0  Exp-Model | 1e+6 | 52.5e-9 | 1e+4 | 0 |
|  | 1e+6 | 0.01 | 1.168e-6 | 3e-9 | 4.32 |  |

CSV R2

$Resistance of third gas switch TOPBRANCH

Trline Linear 17e-9 2.16

Pisection 1e+4 0 1e+6 35e-9 1e+4 1.06e-9 Variable R2 Exp-Model

1e+6 1.67e-3 1.185e-6 3e-9 4.32

CSV R2

$Resistance of fourth gas switch TOPBRANCH

|  |  |  |  |
| --- | --- | --- | --- |
| Trline | Linear | 12e-9 | 2.16 |
| Trline | Linear | 10.6e-9 | 2.16 |
| Rcground | 1e+4 |  |  |
| Trline | Linear | 22.7e-9 | 2.16 |
| ENDBRANCH | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trline Linear 9.1e-9 3.56 | | |  | |
| Trline Linear 9.1e-9 4.06 | | |
| Trline Linear 9.1e-9 4.39 | | |
| Trline Linear 17.8e-9 4.02 | | |
| Trline Linear 9e-9 11 | | |
| Trline Linear 18.2e-9 4.5 | | |
| Rcground 1e+4 | | |
| Trline Linear 10.6e-9 4.5 | | |
| Trline Linear 7.6e-9 4.77 | | |
| Rcground 4.3 | | |
| CSV VR1 | | |
| $Output voltage | | |
| CSV IR1 | | |
| $Output current | | |
| BRANCH | | |
| Trline Linear 18e-9 15 | | |
| Pisection 1e+6 0 1e+6 | | | 400e-9 | 1e-3 0 |
| Variable R2 Exp-Model | | |  |  |
| 1e+6 0.12 958e-9 | | | 2e-9 | 8.49 |
| CSV R2 | | |  |  |
| $Resistance of fifth gas switch | | |  |  |
| BRANCH | | |  |  |
| Rcground 1e+6 200e-12 | | |  |  |
| BRANCH | | |  |  |
| Rcground | 1e+6 | 720e-12 | | |
| BRANCH |  |  |  |  |
| Rcground | 1e+6 | 800e-12 | | |
| BRANCH |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Trline | Linear | 9.1e-9 | 3.56 |
| Trline | Linear | 9.1e-9 | 4.06 |
| Trline | Linear | 9.1e-9 | 4.39 |
| Trline | Linear | 27e-9 | 4.02 |
| Trline | Linear | 15.2e-9 | 4.5 |
| Rcground | 1e+4 |  |  |
| Trline | Linear | 10.6e-9 | 4.5 |
| Trline | Linear | 7.6e-9 | 4.77 |
| Rcground | 4.3 |  |  |

CSV VR1

$Output voltage of side branch CSV IR1

$Output current of side branch

### PBFA II Convolute Model

The PBFA convolute model has three layers of voltage addition to the main branch. The listing of the SCREAMER input file is given below. Notice that perveance-based MITL’s are used, as well as a plasma opening switch. A 10  resistor represents the diode. Voltage sources are specified by tables of voltage versus time. Not all entries for the tables are shown.

PBFA-II Convolute, 8.52MV, RDiode=10, CSV output types only, no user models

Time-step 0.1e-9

Resolution-time 0.075e-9

End-time 200e-9

Number-prints 1

Execute-cycles all

Max-points 501

BRANCH

Voltsource table 1.1 0.0

|  |  |  |
| --- | --- | --- |
| ! | Vmax | Time-delay |
|  | 8.52e6 | 0 |
| ! | Time | Voltage |
|  | 0 | 0 |
|  | 0.6425e-9 | 0.013 |
|  | 2.385e-9 | 0.042 |
|  | (SOME | TABLE ENTRIES OMITTED HERE) |
|  | 104.5e-9 | 0.004 |
|  | 110e-9 | 0 |
|  | 1e-3 | 0 |

Last-entry CSV VOUT

$Source voltage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CSV | POUT |  | | |
| $Source | power |
| CSV | EOUT |
| $Source | energy |
| Trline | Lin | 6.06e-9 | 1.427 | 1.973 |
| Trline | Lin | 0.427e-9 | 11.75 |  |
| Trline | Lin | 0.305e-9 | 20.32 | 14.01 |
| PMitl |  | 1.455e-3 | 2.64e-9 | 5 |
| Adder |  |  |  |  |
| CSV | VIN |  |  |  |
| $Voltage | before | the first add |  |  |
| CSV | VOUT |  |  |  |
| $Voltage | after | the first add |  |  |
| TOPBRANCH | | | | |
| PMitl |  | 34.9e-6 | 0.236e-9 | 11.45 |
| PMitl |  | 392.5e-6 | 1.93e-9 | 10 |
| Adder |  |  |  |  |
| CSV | VOUT |  |  |  |

$Voltage after the second add TOPBRANCH

PMitl 21.6e-6 0.237e-9 17.15

PMitl 149.9e-6 1.1e-9 15

Adder

CSV VOUT

$Voltage after the third add TOPBRANCH

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PMitl | 12.9e-6 | | 0.237e-9 | 25.3 |
| PMitl | 237e-6 | | 1.24e-9 | 20 |
| !PEOS |  | |  |  |
| Rcground | 0.01 | 0 | |  |
| Variable | R1 | PS1\_model | |  |
|  | 65e-9 | 2e-8 100 0.01 | |  |
| CSV R1 | 50e-9 | 200e-9 | |  |
| $Resistance | of the | PEOS | |  |
| PMitl |  | 27.8e-6 0.41e-9 | | 20 |
| !Diode |  |  | |  |
| Rcground | 10 | 0 | |  |
| CSV VR1 |  |  | |  |

$Voltage across the diode CSV IR1

$Current through the diode CSV PR1

$Power in the diode CSV ER1

$Energy in the diode BRANCH

E)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PMitl |  | 775.1e-6 | 1.52e-9 | 5 |
| Trline | Lin | 0.305e-9 | 14.01 | 20.32 |
| Trline | Lin | 0.427e-9 | 11.75 |  |
| Trline | Lin | 6.06e-9 | 1.973 | 1.427 |
| Vendsource table 1.1 0  -8.52e+6 0 | | | | |
| 0 | | 0 | | |
| 0.6425e-9 | | 0.013 | | |
| 2.385e-9 | | 0.042 | | |
| (SOME TABLE ENTRIES OMITTED HER 104.5e-9 0.004 | | | | |
| 110e-9 | | 0 | | |
| 1e-3 | | 0 | | |

Last-entry BRANCH

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PMitl |  | 1.54e-3 | 2.56e-9 | 5 |
| Trline | Lin | 0.305e-9 | 14.01 | 20.32 |
| Trline | Lin | 0.427e-9 | 11.75 |  |
| Trline | Lin | 6.06e-9 | 1.973 | 1.427 |
| Vendsource table 1.1 0  -8.52e+6 0 | | | | |
| 0 | | 0 | | |
| 0.6425e-9 | | 0.013 | | |
| 2.385e-9 | | 0.042 | | |
| (SOME TABLE ENTRIES OMITTED HERE 104.5e-9 0.004 | | | | |
| 110e-9 | | 0 | | |
| 1e-3 | | 0 | | |

)

Last-entry BRANCH

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PMitl |  | 2.22e-3 | 3.18e-9 | 5 |
| Trline | Lin | 0.305e-9 | 14.01 | 20.32 |
| Trline | Lin | 0.427e-9 | 11.75 |  |
| Trline | Lin | 6.06e-9 | 1.973 | 1.427 |
| Vendsource table 1.1 0  -8.52e+6 0 | | | | |
| 0 | | 0 | | |
| 0.6425e-9 | | 0.013 | | |
| 2.385e-9 | | 0.042 | | |
| (SOME TABLE ENTRIES OMITTED HERE) 104.5e-9 0.004 | | | | |
| 110e-9 | | 0 | | |
| 1e-3 | | 0 | | |

Last-entry

### Complex Branches

This sample run deck shows how the maximum level of branches can be called. The run deck has four L2 top branches called from the main branch (L1). Each L2 branch has a single exiting top branch (L3). The L3 branches are listed following the L2 branches in *exactly the order* that they are called in the prior branches. The listing of the SCREAMER input file is given below.

Branch in Branch Test Run Deck

!

! 2014-11-16 RBS

!

! 4 L2 top branches, 4 L3 top branches

!

Time-step 1e-10 Resolution-time 2e-9 End-time 1e-6 Number-prints 5 Execute-cycles all Grids no

Echo-setup no Max-points 1001

!

!Start ciruit definition

!

! Main Branch - Branch #1 BRANCH

RCG 1e+12 1e-6

Initial VC1 50e3

! Cap inductance and ESR Rlseries 0.001 2e-9

TXT VC1

$V\_cap TXT IR2

$I\_in

! Branch #2 location RLseries 1e+12 0.0 TopBranch

! Branch #3 location RLseries 1e+12 0.0 TopBranch

! Branch #4 location RLseries 1e+12 0.0 TopBranch

! Branch #5 location RLseries 1e+12 0.0 TopBranch

!

!Load to ground

!

RCground 0.001 0 TXT IR1

$L1\_Cur(A)

!

! End Main Branch (Branch #1)

!

! Level 2 Branches

!

! Branch #2 Branch

RLseries 0.0 0.0

RLseries 1e+12 0.0

! Call Branch #6 in Branch #2 TopBranch

RCground 0.001 0.0 TXT IR1

$L2\_1\_Cur(A)

!

!Branch #3 Branch

RLseries 0.0 0.0

RLseries 1e+12 0.0

! Call Branch #7 in Branch #3 TopBranch

RCground 0.001 0.0 TXT IR1

$L2\_2\_Cur(A)

!

!Branch #4 Branch

RLseries 1e+12 0.0

! Call Branch #8 in Branch #4 TopBranch

RCground 0.001 0.0 TXT IR1

$L2\_3\_Cur(A)

!

!Branch #5 Branch

RLseries 1e+12 0.0

! Call Branch #9 in Branch #5 TopBranch

RCground 0.001 0.0 TXT IR1

$L2\_4\_Cur(A)

!

! End Level 2 Branches

!

! Start Level 3 Branches

!

! Branch #6 Branch

RLseries 0.004 0.0

RCground 0.001 0.0 TXT IR1

$L3\_1\_Cur(A)

!

! Branch #7 Branch

RLseries 0.001 0.0

RCground 0.001 0.0 TXT IR1

$L3\_2\_Cur(A)

!

! Branch #8 Branch

RLseries 0.004 0.0

RCground 0.001 0.0 TXT IR1

$L3\_3\_Cur(A)

!

! Branch #9 Branch

RLseries 0.001 0.0

RCground 0.001 0.0 TXT IR1

$L3\_4\_Cur(A)

### E-Beam Diode

This sample run deck shows how to model an e-beam diode on a simple single module driver. The listing of the SCREAMER input file is given below.

Electron beam diode test

!

TIME-STEP 0.1E-9 RESOLUTION-TIME 1.0E-9 END-TIME 1E-6

NUMBER-PRINTS 5

EXECUTE-CYCLES ALL

Grids no

ECHO NO

MAX-POINTS 8001

!

!

!

! 2016-06-29 Rick Spielman

!

! A water line driver for brems applications.

!

! MARX: A fast Marx with 32 stages using the GA FMG cap

! 32 x 1.5 µF

!

! IS: A 3 Ohm IS, 60 ns long

!

! electron beam diode: use var resistor

!

!

|  |  |  |
| --- | --- | --- |
| BRANCH |  | |
| RCG | 1E12 | 46.9E-9 |
| INITIAL | VC1 | -2.37E6 |
| TXT VC1 |  | |
| $V\_MARX |
| TXT EC1 |
| $E\_Marx |
| ! |

! Marx inductance and ESR

! 32 x the single capacitor value

! 20 nH and 50 mOhm

!

RLS 0.640 640E-9

!

! Marx switch resistance and inductance

! 16 x 10 mOhm, 16 x 10 nH

!

RLS 0.16 160E-9 TXT IR2

$I\_Marx

!

! The parallel Marx resistance

!

RCG 100.0 0 TXT VR1

$V\_Marx\_out

!

! Marx connection inductance

!

RLS 0 0

!

! \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* INTERMEDIATE STORE \*\*\*\*\*\*\*\*\*\*\*\*\*\*

! 3 Ohm, 30 ns long

!

Trline Lin 30e-9 3 TXT Vout

$V\_IS\_out TXT Eout

$E\_IS\_out

!

! \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* IS Gas Switch \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

!

RLSeries 1e12 50e-9 Variable R2 Decay 1e6 0.1 245e-9 1e-9 TXT IR2

$I\_Switch

!

! Output Water line

!

TRLine Lin 100e-9 3.0 TXT Vout

$V\_TLout TXT Iout

$I\_TLout TXT Pout

$P\_TLout

!

! Load Resistance

!

RCGround 1e6 0.0 VARiable R1 EDIode

!Diode\_type Gap Enhancement Velocity Router 2 2e-2 1.0 0.0 1.25e-2

!

TXT R1

$Z\_load TXT IR1

$I\_Out TXT VR1

$V\_Out TXT PR1

$P\_Out TXT ER1

$E\_Out

!

# User Subroutines

SCREAMER has the ability to incorporate one or more (up to four) user-written subroutines that describe the behavior of variable elements (*R1, C1, R2, L2, R3, C3*) into a SCREAMER run. The following sections of this chapter describe how to create and use your own user subroutines.

## Purpose of User Subroutines

As discussed in **Section 3.3**, SCREAMER provides a "library" of variable element models. There may be cases when this library does not contain a model appropriate for your problem. This is when you would want to write your own subroutine to describe the variable element.

## Format of User Subroutines

User subroutines must be written in Fortran. If there are values in your subroutine that you wish to preserve from one call to the next, you must include them in a Fortran *SAVE* statement within the subroutine. The "subroutine" statement must be of the form:

**SUBROUTINE USER*n* (DT, TIME, P1, P2, P3, P4, V1, V2, V3)**

where *n* is an integer in the range 1 < *n* < 4. If you only have one user subroutine to be used for a SCREAMER run, you may simply specify "*SUBROUTINE USER*" without the *n* qualifier. The *n* qualifier becomes important if you wish to use more than one user subroutine in a single SCREAMER run. More information about multiple user subroutines is provided in **Section 6.3** of this chapter. *DT, TIME, P1, P2, P3,* and *P4* are single precision real variables sent by SCREAMER as input to your subroutine. *V1, V2,* and *V3* are single precision real variables that you return to SCREAMER which describe the behavior of the element you are modeling. You may choose any name you wish for any of these variables, but in the following, it is assumed that they are named as shown.

*DT* is the time-step and *TIME* is the current problem time at the half-time step, *th = (t0 + tn)/2*. The other input parameters are defined as (using **Figure 1** conventions):

For a shunt resistor or capacitor at node i:

* *P1 = Ii-1*
* *P2 = Ii*
* *P3 = IB*

*i*

* *P4 = Vi*

For a series resistor or inductor at node i:

* *P1 = Vi*
* *P2 = Vi+1*
* *P3 = Ii*
* *P4*, *not used*

The returned variables are defined as follows:

For a shunt resistor:

* *V1 = calculated value of Gi*
* *V2*, *not used*
* *V3*, *not used*

For a series resistor:

* *V1 = calculated value of Ri*
* *V2*, *not used*
* *V3*, *not used*

For a shunt capacitor:

* *V1 = calculated value of Ci*
* *V2 = calculated value of dCi /dt*
* *V3 = calculated value of ∂CiVi /∂Vi*

For a series inductor:

* *V1 = calculated value of Li*
* *V2 = calculated value of dLi /dt*
* *V3 = calculated value of ∂LiIi /∂Ii*

If you do not mind generating an incorrect energy check, you need not calculate and return the values *Ci* and *dCi /dt* (shunt capacitor), and *Li* and *dLi /dt* (series inductor). Only the values *∂CiVi*

*/∂Vi* for the shunt capacitor and *∂LiIi /∂Ii* for the series inductor are needed by SCREAMER to calculate the voltages and currents at the next time step.

After you have finished editing your Fortran user subroutine, you should compile it using the Fortran compiler on the system you are using. This will ensure you have no syntax errors in

your file. On the UNIX version of SCREAMER, the subroutine will be compiled and linked automatically to create a new SCREAMER executable.

## Instructing SCREAMER to Use Your Subroutines

In order to instruct SCREAMER to use your user subroutine(s), you must include a user model variable element line in your input deck immediately after the block in which the variable element occurs. The format of the user model variable element line is as follows:

Format: *Variable element* Usn*-Model*

*element* specifies which element is modeled by your user subroutine. It may be *R1*, *C1*, *R2*, *L2*, *R3*, or *C3*. The *n* qualifier tells SCREAMER which user subroutine to use for this element. The value you specify for *n* must correspond to the actual name of your user subroutine. For example, if you wanted to model *R1* using *SUBROUTINE USER2*, your user model variable element line would be:

**Variable R1 Us2-Model**

Note that if you named your subroutine *SUBROUTINE USER* (because you only have one user subroutine), the format of the line would be:

**Variable R1 User-Model**

The following table summarizes the connection between the user model variable element lines in your input deck and your user subroutine names:

"User Model" Line User Subroutine Name *Variable element Us1-Model SUBROUTINE USER1 Variable element Us2-Model SUBROUTINE USER2 Variable element Us3-Model SUBROUTINE USER3 Variable element Us4-Model SUBROUTINE USER4 Variable element User-Model SUBROUTINE USER*

## User Variables

### User Variables Defined

It is conceivable that you will want to calculate values for variables within your user subroutine other than just those which are required by SCREAMER. These variables are referred to as user variables. SCREAMER has the ability to track up to 10 of these user variables for each user subroutine. The word "track" in this context means that you may request that any of your user variables be output using any of the SCREAMER output formats (*TABLE, FILE, TXT, PFF, CSV*).

### Modifying your User Subroutines

In order to enable SCREAMER to track your user variables, you must include the following lines in the declaration portion of your user subroutine:

**REAL U1, U2, U3, U4, U5, U6, U7, U8, U9, U10**

**COMMON /USERVAR/ U1, U2, U3, U4, U5, U6, U7, U8, U9, U10**

The variables *U1, U2, ..., U10* are used to store the values of your user variables. Although SCREAMER can track up to 10 user variables per user subroutine, you need not provide values for all 10 variables. If you wish to only track one user variable, simply provide a value for *U1*. If you wish to track two user variables, provide values for *U1* and *U2*. No matter how many user variables you wish to use, you must include all 10 of them in the *USERVAR* common block.

In order to request that a user variable be output (*TABLE, FILE, TXT, PFF, CSV*) by SCREAMER, include an output request in your input file following your user model variable element line. (See **Section 5.4** for output request formats.) The *Parm* keyword for your user variable output requests is of the form *Un*, where *n* is an integer in the range *1 < n < 10*.

For example:

**CSV U1**

causes user variable *U1* will be saved in a comma separated variable file.

You may assign labels to your user variables for output requests, if you wish. You do not have to provide explicit labels; SCREAMER will create them for you. However, if you wish to, you must carefully follow the guidelines for creating label names outlined in **Section 6.4.3**. Also discussed in **Section 6.4.3** are the default labels which SCREAMER will create for user variables if you do not provide them.

In order to explicitly provide labels for your user variables, include a *ULABEL* line following your output request in your input deck. The format of the *ULABEL* line is:

Format: *Ulabel 1st\_word\_of\_label 2nd\_word\_of\_label*

For example, if you were requesting *PFF* output for user variable *U1*, and you wanted the label to be "Speed m/s", you would include the following two lines in your input file following the appropriate user model variable element line:

**PFF U1**

**ULABEL Speed m/s**

### Labels for User Variables

All "non-user" output labels in SCREAMER are built internally using a consistent naming scheme. To illustrate this naming scheme, the following table lists several SCREAMER variables and their corresponding variable labels:

SCREAMER Variable Name Label Assigned by SCREAMER

*R1 R1 ohms*

*L2 L2 henrys*

*VR2 VR2 volts*

*EC1 EC1 joules*

*QSRC QSRC coul*

*PIN PIN watts*

*IOUT IOUT amps*

To ensure consistency, you should provide labels for your user variables whose formats are the same as those built internally by SCREAMER.

As you can see, the variable label is composed of two words. The first word describes the variable and the second word describes the unit of measurement. Both words, including the space between, must be eleven characters or less.

If you opt not to provide SCREAMER with labels for your variables, the default labels created by SCREAMER for user variables have the form:

U*n* Unknown

where *n* is the number of the user variable.

## Example User Subroutines

This section lists an example of a user routine. It is included with the SCREAMER installation files.

### Gas Switch Model

subroutine user (timestep, time, p1, p2, p3, p4, v1, v2, v3) c

c Gas switch user subroutine for SCREAMER.

c Variable series resistor, depends only on time. c

parameter (ropen = 1e+6) parameter (rclose = 0.1) parameter (tswitch = 1.108e-6) parameter (tau = 1.0e-8)

parameter (zswitch = 6.05) parameter (rtau = 1.0 / tau)

c

c Set the resistance as v1: v2 and v3 are dummy. c

if (time .le. tswitch) then v1 = ropen

else

expa = exp ((-time+tswitch) \* rtau)

v1 = zswitch\*expa / (1.0 - expa + zswitch\*1.0e-6) + rclose end if

c

return end

1. **SCREAMER Output**
   1. **The SCREAMER Log File**

Every time SCREAMER is executed, a SCREAMER log file is created. This log file contains the results of parsing the input deck, as well as a running energy balance check on the circuit.

### Setup Parameter Summary

A summary of the setup parameters, specified in the input deck, is included in the log file. It includes the following information: (1) the time step, (2) the default resolution time for transmission lines, (3) the problem end time, (4) the number of times to list the circuit status in the log file, (5) the number of cycles (time steps) to be executed, and (6) the maximum number of points to store for output requests.

Following is an example of the Setup Parameter Summary section of a SCREAMER log file:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Marx model, 5ns time step

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Time step: 5.000E-09 Default Res-time: 5.000E-09 End time: 1.000E-06 Number of prints: 5

max-pnts stored : 500 Execute all cycles

### Listing of User Circuit

The listing of the user circuit is given in the log file. This section echoes the circuit which you described in your input file in a nice format. The branch and block numbers assigned by SCREAMER are included for your information. If any errors are found in your input file, the lines in error will be flagged, and execution of the current SCREAMER run will be halted after the input file has been completely read and echoed. If one or more errors are found in your input file, the following line will appear in the SCREAMER log file:

### Errors found in data file, execution halted.

Following is an example of the user circuit listing section of a SCREAMER log file:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Listing of the User Circuit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\* Branch 1 \*\*\*\*\*\*\*\*\*\*\*\* Block 1 : pi section.

R1= 1.000E+12 C1= 2.200E-08 R2= 2.500E+00 L2= 1.200E-05 R3= 1.400E+03 C3= 1.600E-08

Initial condition: Voltage on C1 = 5.000E+06

PFF Output Request for block 1. Variable: Voltage across C1 Values on the whole time step will be used.

PFF Output Request for block 1. Variable: Voltage across R3 Values on the whole time step will be used.

PFF Output Request for block 1. Variable: Current in R3 Values on the whole time step will be used.

PFF Output Request for block 1. Variable: Power Dissipated in R3 Values on the whole time step will be used.

PFF Output Request for block 1. Variable: Energy Dissipated in R3 Values on the half time step will be used.

File read with no errors, continue execution.

### The Initial Circuit Status

The energy balance of the circuit at the initial time is included in the log file. This section will always be included in the log file, regardless of how many execute cycles are selected or how many prints of the circuit status are selected.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Initial SCREAMER Circuit Status

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Time = .000E+00 Cycle = 0

|  |  |
| --- | --- |
| Energy from all sources: | 2.750E+05 |
| L\*I\*I/2 energy stored in inductors: | .000E+00 |
| C\*V\*V/2 energy stored in capacitors: | 2.750E+05 |
| G\*V\*V energy dissipated in shunt resistors: | .000E+00 |
| R\*I\*I energy dissipated in series resistors: | .000E+00 |
| Ldot\*I\*I/2 energy in variable inductors: | .000E+00 |
| Cdot\*V\*V/2 energy in variable capacitors: | .000E+00 |
| Relative error in energy sum: | .000E+00 |

### Circuit Status at Specified Intervals

The circuit energy balance at specified intervals is included in the log file. The number of times that this information is printed is determined by dividing the total number of time steps by the number of prints requested in your input file + 1. For example, if the end-time specified in your input file was 1 µsec and the time step specified was 5 ns (200 time steps), and you requested 5 prints, the energy balance would be printed 6 times -- first at time 0, then at time steps 40, 80, 120, 160, and 200. The problem time given at each printed time step is the time at the half- step.

The energy balance shows how energy is currently distributed in the problem and how well the energy balance has been maintained between the energy produced by the sources and the energy dissipated or stored in the circuit elements. The energy from all sources is that produced by all source blocks and that initially placed into the circuit via an initial condition card. The following listing is an example of an energy check in a SCREAMER log file:

Time = 1.975E-07 Cycle = 40

|  |  |
| --- | --- |
| Energy from all sources: | 2.750E+05 |
| L\*I\*I/2 energy stored in inductors: | 3.465E+04 |
| C\*V\*V/2 energy stored in capacitors: | 2.393E+05 |
| G\*V\*V energy dissipated in shunt resistors: | 7.220E+00 |
| R\*I\*I energy dissipated in series resistors: | 1.044E+03 |
| Ldot\*I\*I/2 energy in variable inductors: | .000E+00 |
| Cdot\*V\*V/2 energy in variable capacitors: | .000E+00 |
| Relative error in energy sum: | 1.110E-04 |

The *LI2 /2* and *CV2 /2* terms (2nd and 3rd terms) are the usual energy stored in the inductors and capacitors. The *GV2* and *RI2* terms (4th and 5th terms) are the energy dissipated in the shunt and series resistors. For nonlinear *L* or *C*, the *(dL/dt)I2 /2* and *(dC/dt)V2 /2* terms (6th and 7th terms) are the energy used to change the value of the inductance or capacitance. The relative error in the energy balance is difference of the sum of terms 2-7 and term 1 (the energy from all sources), divided by term 1.

### Execution Summary

The last section in the log file is a summary of the run. It displays the name of the input file, the date and time, the execution time (wall-clock time, not "CPU time") and the version of SCREAMER used. The following listing is an example of this section.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Simulation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The SCREAMER input file used: marx Current time: 16:26:39

Current date: 19-Jul-95 Run time: 2 seconds SCREAMER version 2.0

## Circuit Parameter Output

In addition to the SCREAMER log file, SCREAMER also provides a robust output package for examining circuit parameters in a variety of formats. Circuit parameter output is not automatic. You must specify the output you wish SCREAMER to produce by including output requests cards in your input file. SCREAMER currently provides circuit parameter output in these different

formats: *FILE, TABLE, TXT, PFF, CSV, SFC*. The following sections provide detailed information and examples of each output format provided by SCREAMER.

### FILE

If your SCREAMER input file contains any *FILE* output requests, one ASCII (text) file for each request will be created. The first line in a file created using a *FILE* output request contains two values: (1) the number of time vs. parameter pairs listed in the file and (2) the name of the parameter, as specified in the title card following the *FILE* output request in your input file. If you do not provide a title identifying the parameter, SCREAMER will create a name for you. This name will contain the branch number, block number, and type of block for the selected parameter. For example, if the *FILE* output request was for the parameter VC1 in a -section block, in the first branch of the circuit, the label SCREAMER provides is:

**Brn 1, Blk 1 : pisect {VC1 volts}**

The remainder of the file is simply a listing of the time vs. parameter coordinates, one pair per line.

A file created using a *FILE* output request is generic in the sense that its format was not designed specifically to be used as input into any one analysis package. However, with slight modification, a file created using a FILE output request can be used as input to a variety of plotting and/or analysis packages.

The naming convention for the files created by *FILE* output requests is to append the characters ".f*n*" to the name used for the input file where "*n*" is a number running from 1 to the number of *FILE* output requests. For example, if the name of the input file is "test", the third *FILE* output request would be placed in the file "test.f3". Following is an example of a file created as a result of a *FILE* output request with some of the lines omitted for brevity:

41 OUTPUT VOLTAGE 0.000E+00 0.000E+00 2.250E-08 6.657E+03 4.750E-08 2.929E+04

(SOME LINES OMITTED HERE) 9.475E-07 5.292E+06

9.725E-07 5.340E+06

9.975E-07 5.374E+06

### TABLE

If your SCREAMER input file contains any *TABLE* output requests, one ASCII (text) file for each request will be created. The first line of a file created using a *TABLE* output request is the title of the current SCREAMER problem. The second line is the label of the parameter being examined, as specified in the title card following the *TABLE* output request in your input file. If you do not provide a title identifying the parameter, SCREAMER will create a unique label for you. See **Section 7.2.1** for more information on SCREAMER’S naming convention for parameter

labels. The third line indicates how many time vs. parameter pairs are listed in the file. The fourth line contains two values: the scale factor for which all parameter values must be multiplied by to get their actual values, and a delay factor. The remainder of the file (except the last line) is simply the time vs. parameter coordinates, one pair per line. The last line of the file is the string "Last Entry."

A file created using a *TABLE* output request may be used directly in a subsequent SCREAMER run. Simply "paste" the appropriate *TABLE* file into the desired SCREAMER input file. There are five places in a SCREAMER input file where this would be appropriate: (1) following a beginning-of-main-branch voltage source card, where the function specified is ’*TAB*’, (2) following a end-of-branch voltage source card, where the function specified is ’*TAB*’, (3) following a beginning-of-main-branch current source card, where the function specified is ’*TAB*’, (4) following a end-of-branch current source card, where the function specified is ’*TAB*’, and (5) following a resistor table model card. See **Sections 5.3.2** and **5.3.4** for more information.

The naming convention for the files created by *TABLE* output requests is to append the characters ".t*n*" to the name used for the input file where "*n*" is a number running from 1 to the number of *TABLE* output requests. For example, if the name of the input file is "test", the second *TABLE* output request would be placed in the file "test.t2". An example of a file created as a result of a *TABLE* output request is listed below. Notice that each of the first three lines of the file are preceded by an exclamation point. This will cause these lines to be treated as comments if your opt to include the file in a subsequent SCREAMER input file. Again, some lines have been omitted for brevity.

!Marx model, 5ns time step

!OUTPUT VOLTAGE

! 101 points in the table 5.374E+06 0.0

0.000E+00 0.000E+00

7.500E-09 1.513E-04

1.750E-08 7.558E-04

(SOME LINES OMITTED HERE) 9.775E-07 9.952E-01

9.875E-07 9.978E-01

9.975E-07 1.000E+00

Last-entry

### TXT

If your SCREAMER input file contains any *TXT* output requests, one file for all *TXT* requests will be created. This file may serve as input to the *TXT* graphics package or to some spreadsheet packages. Note that a title card following a *TXT* output request in your input file is not used by SCREAMER; it is simply ignored. Note: as of V4.3.3 *TXT* is the preferred output request for a single ASCII file. The older UFO output request is still supported for backward compatability of older run files. The UFO output request will be deprecated time in the future.

The naming convention for the file created by *TXT* output requests is to append the characters "\_d.txt" to the name used for the input file. For example, if the name of the input file is "test", the *TXT* output requests would be placed in the file "test\_d.txt".

### PFF

If your SCREAMER input file contains any *PFF* output requests, one file for all *PFF* requests will be created. This file may serve as input to any program that was written to process *PFF* files, such as PFIDL. Note that you may transfer this file to any other computer on which the *PFF* library has been used to write programs that read *PFF* files, without translation. See the *PFF User’s Guide* for information about using *PFF* files. Note that SCREAMER will not overwrite existing *PFF* files, so if you wish to rerun a simulation without renaming the input file, you must first delete the corresponding *PFF* file.

The naming convention for the file created by *PFF* output requests is to append the characters ".pff" to the name used for the input file. For example, if the name of the input file is "test", the *PFF* output requests would be placed in the file "test.pff".

### CSV

If your SCREAMER input file contains any *CSV* output requests (comma separated variables), one file for all *CSV* requests will be created. This file may serve as input to any spreadsheet program that reads "comma-separated variable" formatted files, such as *EXCEL*.

The naming convention for the file created by *CSV* output requests is to append the characters ".csv" to the name used for the input file. For example, if the name of the input file is "test", the *CSV* output requests would be placed in the file "test.csv".

Following is an example of a file created as a result of *CSV* output requests. In this example, four *CSV* output requests were made in the SCREAMER input file. Again, lines have been omitted for brevity:

time,SOURCE CAPAC,OUTPUT VOLTA,OUTPUT CURRE,OUTPUT POWER,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .000E+00, 5.000E+06, .000E+00,  2.500E-09, 5.000E+06, 1.626E+02, (SOME LINES OMITTED HERE) | | | .000E+00,  1.162E-01, | .000E+00,  1.890E+01, |
| 9.875E-07, | 1.018E+06, | 5.362E+06, | 3.830E+03, | 2.054E+10, |
| 9.925E-07, | 1.012E+06, | 5.368E+06, | 3.835E+03, | 2.058E+10, |
| 9.975E-07, | 1.007E+06, | 5.374E+06, | 3.839E+03, | 2.063E+10, |

### SFC

If your SCREAMER input file contains any *SFC* output requests, one file for all *SFC* requests will be created. This file may serve as input to programs such as *EXCEL* or *XMGR*.

The naming convention for the file created by *SFC* requests is to append the characters ".sfc" to the base name of the input file. For example, if the name of the input file is "test", the *SFC* output requests would be placed in the file "test.sfc".

Following is an example of a file created as a result of SFC output requests. In this example, four SFC output requests were made in the SCREAMER input file. Again, lines have been omitted for brevity:

TITREG=Marx model, 5ns time step DATHEU=01/27/99

TYPEDO=REEL NBCOLO= 5

NBLIGN= 201

TITCOL=Time ;CAPACITOR VOLTA;OUTPUT VOLTAGE ;OUTPUT CURRENT

;OUTPUT POWER ;

LABCOL=Time ;Voltage ;Voltage ;Current

;Power ;

UNICOL=Seconds ;Volts ;Volts ;Amps

;Watts ;

COMENT=SCREAMER 2.2 DONNEE=

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .000000E+00 5.000000E+06 .000000E+00  5.000000E-09 4.999764E+06 3.252968E+02 (SOME LINES OMITTED HERE) | | | .000000E+00  2.323549E-01 | .000000E+00  7.558430E+01 |
| 9.900000E-07 | 1.014768E+06 | 5.365374E+06 | 3.832410E+03 | 2.056231E+10 |
| 9.950001E-07 | 1.009610E+06 | 5.371269E+06 | 3.836620E+03 | 2.060752E+10 |
| 1.000000E-06 | 1.004870E+06 | 5.376587E+06 | 3.840419E+03 | 2.064834E+10 |

## Interactive Input/Output

SCREAMER will send output to the default window as it is running. Run the sequential, 64-bit version Screamer from a terminal window by typing the command to run Screamer followed by the file name of the run deck as shown below. When Screamer is invoked in the terminal window it assumes a filename follows the program call. The placement of the file name on the same terminal window line as the program call allows the simple use of batch files to run a large number of cases with no interruptions. No further user input is needed to run Screamer.

When run, Screamer provides a range of information on the program itself (version number), date and time, the local computer information, the run file, the status of the run file inout, and the number of nodes of the problem (effectively the memory requirement). A sample run is provided below. In this case, Screamer is run from the termainal window of a MacBook Pro and the run file follows the call to run Screamer.

RBS-MacBook-Pro:run\_decks rbspielman$ ./screamer64 TL\_test\_50ohm.txt

\*\*\*\*\* SCREAMER v4.4.1 \*\*\*\*\*

\*\*\*\*\* The current time is: 20:05:55:021 \*\*\*\*\*

\*\*\*\*\* The current date is: 12/09/2019 \*\*\*\*\*

Host computer = RBS-MacBook-Pro.local User name = rbspielman

Current Working Directory =

/Users/rbspielman/Documents/code/fortran/screamer/release\_4.4.1\_beta/run\_decks

Screamer input file name = TL\_test\_50ohm.txt Input file read with no errors.

Total nodes in simulation = 203 Time = 0 ns

SCREAMER is now writing to the output file Run time: 0.445 seconds

Done

RBS-MacBook-Pro:run\_decks rbspielman$

When SCREAMER begins the circuit simulation and, as it runs, displays the current simulation time as regular intervals:

Upon completion, SCREAMER creates the files containing the output requests, shows the execution time (wall-clock time for the simulation).

Similarly, SCREAMER can run a multi-core, 64-bit version SCREAMER from a terminal window by typing the command to run SCREAMER followed by the file name of the run deck as shown below. In this case, the executible is “screamerp” When SCREAMER is invoked in the terminal window it assumes a filename follows the program call AND the number of threads to be used in the solution. The placement of the file name and the thread number on the same terminal window line as the program call allows the simple use of batch files to run a large number of cases with no interruptions. No further user input is needed to run SCREAMER.

As before, when run, SCREAMER provides a range of information on the program itself (version number), date and time, the local computer information, the run file, the status of the run file inout, and the number of nodes of the problem (effectively the memory requirement). A sample run is provided below. In this case, SCREAMER is run from the termainal window of a MacBook Pro, the run file follows the call to run SCREAMER, and the number of threads follows the file name.

RBS-MacBook-Pro:run\_decks rbspielman$ ./screamer64p TL\_test\_50ohm.txt 2

\*\*\*\*\* SCREAMER v4.4.1 \*\*\*\*\*

\*\*\*\*\* The current time is: 20:36:00:732 \*\*\*\*\*

\*\*\*\*\* The current date is: 12/09/2019 \*\*\*\*\*

Host computer = RBS-MacBook-Pro.greenlightnetworks.com User name = rbspielman

Current Working Directory =

/Users/rbspielman/Documents/code/fortran/screamer/release\_4.4.1\_beta/run\_decks

Screamer input file name = TL\_test\_50ohm.txt # of cores (logical) available = 8

# of threads used = 2

Input file read with no errors. Total nodes in simulation = 203 Time = 0 ns

SCREAMER is now writing to the output file Run time: 1.617 seconds

Done

Note: The following floating-point exceptions are signalling: IEEE\_UNDERFLOW\_FLAG IEEE\_DENORMAL

RBS-MacBook-Pro:run\_decks rbspielman$

If you place a copy SCREAMER in /usr/local/bin/ with the cp command (It is likely you will have to type “sudo cp screamer64 /usr/local/bin/” and enter your password at the prompt.) Then simply typing “screamer64” anywhere will run SCREAMER. Remember to put the run deck in the same location as where you are running SCREAMER.

SCREAMER is designed to be run in large batch files. To do this simply create a batch.bat file and type all of the SCREAMER runs that are desired a line at a time. Each run must have its own runfile.txt associated with it. This is incredibly useful to run cases with a wide variation in parameters.

## File Naming Conventions in SCREAMER

SCREAMER output file names are generated using the basename of the SCREAMER input file. SCREAMER versions 2.2 and beyond use a different convention for naming output files than previous versions. This was done to accommodate the widely used practice of adding the suffix “.dat” for SCREAMER input files. Previously, the entire filename of the SCREAMER input file was used to generate all output files. Now, only the characters to the left of the rightmost ‘.’ character are used to generate the output filenames. The following examples, for a SCREAMER input file using only TXT output requests, illustrate the filenaming methods for the current version.

### Current Method

|  |  |  |
| --- | --- | --- |
| **input file name** | **log file name** | **TXT file name** |
| test.dat | test.log | test\_d.dat |
| x1.input | x1.log | x1\_d.txt |
| Pflow.txt | pflow.log | pflow\_d.txt |

# Installation and System Dependent Information

This chapter contains installation procedures and system-dependent information about the LINUX (UNIX), Macintosh and PC versions of SCREAMER. Screamer has unified source code for all platforms.

## LINUX (and UNIX)

### Installation

First, make sure that the desired version of LINUX is installed on the computer. If necessary update the version with the standard LINUX update command. Second, you must install the GNU gcc package. Check the version with >gcc -v or >gfortran -v. Install with:

sudo apt-get install gcc

sudo apt-get install build-essential sudo apt-get install gfortran

To install SCREAMER, create a directory in which to install the files. Modern LINUX systems have a desktop so you simply click to the desired folder and insert a new folder. Copy the SCREAMER files for the desired release to that new directory from the SCREAMER web site or external media, e.g. memory stick.

The SCREAMER release will have two subdirectories: "src", "run\_decks". "src" contains all the source files, header files and the "batch file". "run\_decks" contain many sample input decks.

In order to create the executable and object files, you may need to modify the "batch file" for your particular system. The path to the locations of the required libraries may (or may not) require a specific path to be included. These lines describe, respectively, Fortran compiler options, loader/linker options, and a list of the libraries needed.

Then run the “batch file with ./screamer64.bat and an executable will be generated and be placed in the src folder and it will make a copy of the executable in the “run\_decks” sub- directory. Go to the bash (terminal) window and cd to the “run\_decks” sub-directory and type

./screamer64, then enter the run deck name, and type return. You are done for the sequential version.

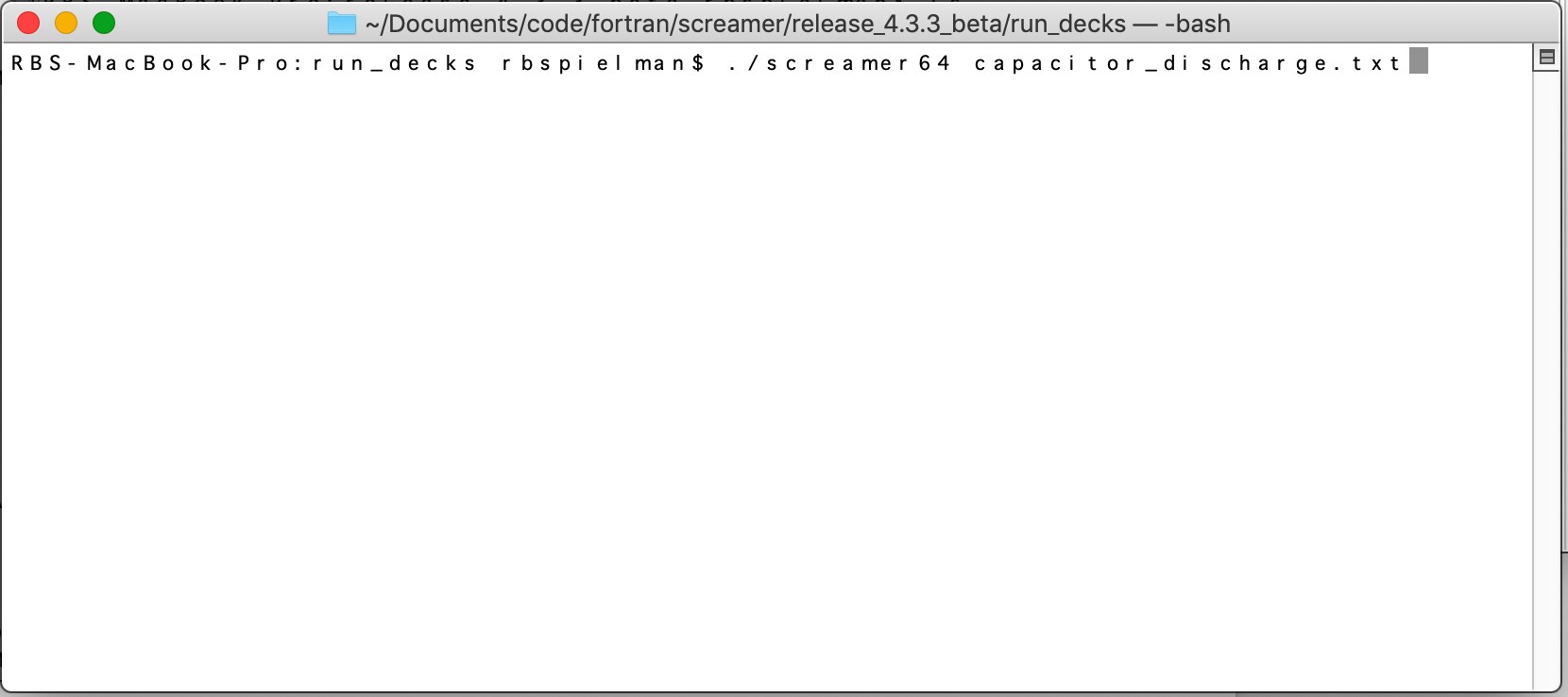
### Obtaining Sample Input Files and Sample User Subroutines

Many sample input files are included in the "run\_decks" directory.

*marx* (input deck describing a simple Marx generator circuit) *demon* (input deck describing the DEMON accelerator) *pbfa2conv* (input deck describing the PBFA II vacuum stack) *maguser.f* (user-written subroutine example for a magnetic switch) *gasuser.f* (user-written subroutine example for a gas switch)

### Execution

SCREAMER is typically executed from a bash window as shown below.



In the bash window one enters ./screamer64 followed by the run file name on one line to run the sequential version of SCREAMER.

## Macintosh OS

SCREAMER is built to run on the latest Macintosh operating systems. As of December 2019 SCREAMER is fully compatible with the Mojave (macOS 10.14.6) operating system. We strongly recommend that all users upgrade to the latest macOS operating system as there is no assurance that SCREAMER will continue backwards compatibility as the gcc compilers and the macOS change in the future. Note: macOS Catalina (10.15) breaks past 32-bit applications and many users are reluctant to upgrade their Macs at this time.

### Installation

The Macintosh can be treated like any other Unix or Linux installation. Following the instructions in **Section 8.1** will generally work for the Macintosh OS as well. In this case, the executable can be run from the standard terminal window.

Please see **Chapter 13** for detailed compilation instructions. As of SCREAMER version 3.2.4.2 SCREAMER can be compiled by GNU gFortran, Intel Fortran, IBM Fortran, and Absoft Fortran. The fastest compiler, as of November 2018, for the Intel processor is the Intel Fortran compiler (~ 1.5X faster).

### Execution

Open up a terminal or bash window (macOS, LINUX, UNIX, Windows) and cd to the run deck directory. The SCREAMER input data file name (run deck) does not need to be in the same folder as the SCREAMER executable however, the full path to the run deck must be included in the

name of the run deck. The output file will be created in the directory with the executable. For analyzing output from SCREAMER, it is recommended that the TXT or CSV formats be used. These data files are compatible with Excel, Kaleidagraph, and Deltagraph. Other open source analysis tools are also available.

### Output:

All output formats can be generated by the Macintosh version of SCREAMER.

### User-written Subroutines:

The Macintosh version of SCREAMER does not have an automated capability to include user- written subroutines that describe variable circuit elements. This can be accomplished, however, by linking a new version of SCREAMER in which the user’s subroutine is linked, rather than the placeholder routines of the same name. These subroutines are named "user", "user1", "user2", "user3" and "user4" and are located in the files "usrfunc.f", "usr1func.f", "usr2func.f", "usr3func.f" and "usr4func.f", respectively. That is, in the SCREAMER source code there are Fortran statements such as "call user", "call user1", etc.

NOTE: Such a user-created version of SCREAMER, which could conceivably include user- written versions of all 5 subroutines, can be used in place of the base version of SCREAMER. It will still have the same capabilities of the base version.

## Windows PC OS

The Windows 10 version of SCREAMER is is now identical to the macOS and LINUX versions. Microsoft has fully implemented LINUX into Windows 10 (not emulation).

### Installation

SCREAMER is installed in a tedious but straightforward manner by installing the Bash application on Windows first. After Bash on Windows is installed, you just install screamer from bash as you would do normally from Linux.

"Bash on Windows provides a Windows subsystem and Ubuntu Linux runs atop it. It is not a virtual machine or an application like Cygwin. It is a complete Linux system inside Windows

Step-by-step instructions:

1. Install Bash on Windows. There are several different options available. I used Ubuntu from Microsoft Store. Follow this step How to Install Linux Bash Shell on Windows 10.
2. Start installed Ubuntu App (or other Bash on Windows you installed). It should be accessible from Windows Start Menu, or Taskbar, or from wherever it was installed.

At this point to build screamer simply follow Ubuntu Linux notes from iac.isu.edu/screamer.html.

1. From Bash on Windows App

3a) Install All GNU gcc Libraries just as in the LINUX instructions:

$sudo apt update (just to be sure your LINUX is up to date.)

$sudo apt install gcc

$sudo apt install build-essential

$sudo apt install gfortran 3b). Check Them All:

$gcc -v

$gfortran -f

1. Go back to Windows and download latest screamer release from iac.isu.edu/screamer.html.
2. Go back to Bash App and copy the download release to your Bash Shell folder.

$cp /mnt/c/path\_where\_downloaded . (the dot represents current dir where the file will be copied)

1. Unpack download

$tar -xzvf release\_4.3.tar.gz

1. Build screamer

7a) Go to the source folder

$cd /src

7b) if "allocation memory error" or "similar errors" occur during the compiling process, reduce the size of max\_branches, max\_nodes as needed (in file zdemmax.h). To make screamer run on a i3-2310 4GB laptop zdemmax.h was modified as follow:

* max\_branches = 65 (default value 250)
* max\_nodes = 10,000 (default value 40,000. Checked with 15,000 not work) 7c) build screamer executable

$./screamer64.bat

### Execution:

Follow the LINUX and macOS instructions. In the bash window simply type.

$./screamer64 filename.txt

### Output:

Output is default placed in the directory where Screamer is executed. You are free to change the location of Screamer and the output path.

1. **OpenMP SCREAMER**

As of Version 4.4 a multi-core, multi-thread version of SCREAMER has been created. The main solver routines have been totally rewritten to allow the ability to run SCREAMER on multi-core computers. In addition, the overall solver efficiency has been improved by > 2X.

The parallel version of SCREAMER must be separately compiled usind the screamer64p.bat file. A separate executible, screamer64p, will be created. This code requires user inputs for the number of threads to be used. A typical command line would be:

$ screamer64p filename.txt #thread

where in this case #thread is an integer less or equal to the number of logical cores of the computer being used.

The SCREAMER array architecture is designed to use multiple branches and multiple levels of branches. It is NEVER a good idea to have the number of threads larger than the number of L2 and L3 branches since it is only these branches that benefit from parallel operation.

Use of multi-thread, multi-core operation in SCREAMER is not suggested at this time unless: 1) multiple identical branches are called and 2) those branches are very large (> 4,000 nodes). This sort of problem is commonly found with large multi-module machines. For example, a machine with 36, parallel modules could be modeled with a run deck of 1 main (L1) branch and 36, L2 branches, The small L1 branch contains the common load information. Even in this case, multi-core solutions only add value if the individual L2 branches are very large in the number of nodes (>4000). This typically only happens when transmission lines are highly resolved.

The multi-core version of SCREAMER will be slower than the sequential version for most cases, at this time. This is due to the fact that the time to create the thread (for each array solution – each time step) can be longer than the solution time for the sub-array. In this case, the sub- array is the array related to the L2 (or higher) branches. (Improvements to the multi-core version of Screamer are ongoing.) It is suggested that comparisons be made between the sequential version of Screamer and the multi-core version of Screamer before making large runs assuming that more threads are better.

For example, a run deck with a single, small L1 branch and 2, very large L2 branches is optimally solved with 2 threads. In this particular case, the thread time (in parallel) and the solution time (in parallel) is faster than running the sequential version even though the thread time is comparable to the solution time. the overall improvement is ~ 20% (as of now). More and longer L2 (and L3) branches will have larger improvements in run time up to the number of cores in the processor.

Finally, much of the slowdown in the speed of multi-core simulations is due to excessive caching of information. Laptop computers will often have a 4 processor CPU with ~ 256 kB of L1 cache (dedicated memory per core) and ~ 8 MB of L2 cache (shared cache). A large

solution array can be ~ 1 GB an would take many cache cycles to get the information to the cores. Future improvements to the main solver will improve the caching slowdown.

The latest (late 2019) Intel processors used in the MacPro and LINUX servers have much improved cache sizes. For example, the W-3245M Intel processor has L1 – 1 MB per core, L2 = 16 MB (1 MB per core directed), and L3 – 22 MG (shared cache). These large caches make parallel operation much more interesting.

# SCREAMER Circuit Parameters and Memory Limits

This chapter summarizes the limits on various code quantities imposed by SCREAMER. Users may override these values by editing the file "zdemmax.h", then recompiling and relinking SCREAMER. NOTE: The memory requirements have increased with Screamer V4.x because a full matrix solver is implemented. Each explicit circuit node in SCREAMER has two words associated with it (current and voltage). The bulk of the memory required, measured in double- precision words (real\*8), used by the main SCREAMER arrays is given by:

Memory (bytes) = [(max\_nodes\*2)2 + (max\_nodes\*2)]x 8 Bytes/word

For a setting of max\_nodes = 10,000 the required memory is ~ 4 GB. We have routinely operated with max\_nodes > 40,000.

SCREAMER naturally accesses all available RAM and disk swap space (available unused disk space) as needed. On the macOS this use of swap space as virtual memory is transparent. In LINUX and Windows 10/LINUX the size of the swap space is set by the size of the swap partition. The swap partion is established at the time LINUX is installed. The suggested minimum RAM is 8 GB and the recommended RAM is 16 GB. (Although SCREAMER will install in as little as 4 GB.) The recommended swap partition size is 60 GB. Unless the size of the problem is large it will run in RAM at full speed. If virtual memory is used then the speed of SCREAMER will be reduced due to the slowness of the resulting page swaps. More RAM is always better.

The default limits on the circuit parameters in SCREAMER V4.3 are given in the following table:

|  |  |
| --- | --- |
| Quantity | Max. Value |
| no. of branches (main+secondary) | 250 |
| no. of circuit blocks + output requests per branch | 300 |
| Maximum no. of circuit nodes | 40,000 |
| no. of output requests | 400 |
| Maximum number of points stored in an output  request (except a TABLE output request) | 20,001 |
| no. points stored for a TAB input waveform and for  a TABLE output request | 1001 |
| no. of variable elements | 65 |
| no. of initial conditions (voltage or current) | no. branches +1 |
| no. of lossless transmission line blocks | 6000 |
| no. of MITL blocks | 100 |
| no. of segments in a transmission line | 6001 |
| Maximum variable elements | 65 |
| Maximum points resolution skin depth | 1001 |
| Maximum number of time steps allowed in skin  depth calculation | 20,001 |

# Debugging Capabilities

These additional settings can be utilized via the input deck to have debugging information printed in the log file. This information is cryptic and requires knowledge of the internal workings of SCREAMER. These lines are not required in the input deck. If they are not entered, default values (recommended settings) will be used.

### Setup Condition Cards:

*Detail-prints level* {*level* is *Min* (default) or *Full*} *Echo ans* {*ans* is *No* (default) or *Yes*}

*Detail-prints* is set to *Min* by default. When set to *Full*, additional circuit information, on a node-by-node basis, is printed to the log file each time the circuit status is printed. If you have a lot of circuit nodes and *Number-prints* is quite high, you can generate a very large log file.

*Echo* is set to *No* by default. When set to *Yes*, an echo of the initial data in SCREAMER’s internal circuit arrays will be printed.

# Problem Size and Run Time

The minimum memory needed to run SCREAMER depends on the details of the circuit. SCREAMER publications describe this in detail but we summarize here. The size of the solver matrix is # nodes\*2 X # nodes\*2. The number of nodes in a run deck depends in detail on the circuit elements.

RLGround – 1 node per call RLSeries – 2 nodes per call

TRLines – 1 node per resolution element (= int (tlength/tres + 0.5)) Branch – 1 additional node

NOTE: The largest number of nodes is often generated by highly resolved transmission lines. A single TRLine call can easily generate > thousand of nodes.

As we saw above, the memory requirements are set primarily by the main solver matrix. For a very common problem containing 5,000 nodes, the main solver matrix requires 800 MB of DRAM.

SCREAMER V4.4.1 delivers several core solver improvements:

1. The core solver has been rewritten with extreme care with regard to speed. A base improvement of ~ 2X is typical.
2. The user has the option of running single core or multiple cores. Calling screamer64 is the single-core version and screamer64p is the parallel, multiple core version. There is NO improvement in run time unless multiple L2 or L3 branches are used. The parallel version

of SCREAMER requires a minimum of 2 branches are used. Even with 2 or more L2 or L3 branches in the problem, the speed up depends on the size of the branch. Significant time is used generating the treads needed for parallel operation. Unless the solver time for one branch is larger than the thread time there will be no speed up and there can actually be a slow down. Given a CPU with many cores, many branches, and very long and complex branches parallel run times can be much shorter.

The run time of any problem is set by the solution time of the solver matrix AND the number of time steps in the problem (total number of matrix solutions). One can reduce the solution time by reducing the problem time and mimimizing the size of the problem matrix.

## Practical Time Step and Resolution Rules

The user has the ability to set an arbitrary time step and, multiple, arbitrary transmission line resolutions. This gives great flexibility and the room to make errors. Here are some good rules- of-thumb to follow.

1. Make sure your time step is at least 2000X smaller than the total run duration set in the run deck. For example, a run duration of 1 µs would typically have a time step less than 1 ns. We usually suggest shorter.
2. The transmission-line resolution is at least 20X smaller than the length of the transmission line. For example, a transmission line having a length of 2 ns should have a resolution time smaller than 100 ps.
3. When transmision lines are used and a transmission-line resolution is specified, the time step MUST be less than 1/20th of the transmission-line resolution to eliminate calculational noise. For example, a transmission line resolution of 100 ps calls for a time step smaller than 5 ps.

These rules, taken together, provide clear guidance for any complex problems. Given a problem with a run duration of 1000 ns, and a shortest transmission-line length of 1000 ps will require the following run deck specifications.

Run duration – 1000 ns

Shortest transmission-line length – 1000 ps Transmission-line resolution – 50 ps

Time step – 2.5 ps

# Versions

SCREAMER version 3.x has significant rewrites to the subroutine structure and the Fortran in an attempt to make the code more portable. SCREAMER 3.0 was ported to the GNU g77 to simplify the ease of cross platform compilation. SCREAMER 3.2.4.x was ported to GNU gFortran after g77 was no longer supported. Most of the changes in SCREAMER 3.x are internal and not apparent to the user. With the port to gFortran and support for the GNU compiler collection at gcc version 4.4, SCREAMER is fully portable within the GNU framework. (As of SCREAMER version 3.2.4.x the code has also been successfully compiled with the Intel Fortran compiler and the IBM compiler.)

Screamer V4.x has replaced the legacy matrix solver with an optimized, generalized solver. In addition, the full 2-D matrix formulation is implemented in order to improve transparency in the solver.

Screamer V4.1.4.1 has implemented improved array handling and no longer has any memory limit. This means that the user is free to increase the maximum number of nodes to any value within the physical memory of the computer.

Please read the changelog.txt in the src directory for a listing of all version changes.

## Source file names

The source files names may not be compatible with the DOS 8 character filename limit. With later versions of the macOS and LINUX there are few limitations on file names Also, all Fortran source files now end with a ".f" suffix and all the header files end with a ".h" suffix.

## Execution status

SCREAMER displays the current simulation time at regular intervals during the run to give an indication of its status towards completion.

## Modifications to circuit blocks and models

### The Lossless and Lossy (MITL) Transmission Line Blocks.

The definition of transmission lines as a series-connection of circuit nodes with shunt capacitors, series inductances and, for lossy lines, variable shunt resistors, has been modified. The -network used for the lossless line was changed to that shown in Figure 4. The racetrack MITL, shown in Figure 5, was similarly changed, except for the presence of shunt resistors. The perveance MITL now uses the same definition as the racetrack MITL. These changes were made because the old versions did not function correctly when multiple lines were connected together. There was always an impedance mismatch at the first and last elements, even though the impedance for the entire line was correct. This slight mismatch at the ends did not affect the transmission line calculations significantly, but it did have an effect on the stability of the

MITL model. With this new setup there is no mismatch at the ends. Multiple, identical impedance lines can be stacked together without any mismatch. These changes are transparent to the user.

### Racetrack MITL model.

This MITL model had some very serious errors which were corrected. First, it did not calculate the perveance correctly. There were two compensating errors that allowed the model to give a nearly correct result for some setups, but it was strongly dependent on the resolution time. That is, by choosing a different resolution time the result could be significantly changed. The other error was related to the transmission line model, which was used to setup the LC network. The conductances at the end elements were not being correctly calculated, and that lead to instability.

These problems have been fixed, although the model still tends to oscillate if the time step is too large. Time step averaging has been added to help damp oscillations. The model has been checked against 2D particle-in-cell simulations and agrees very well.

The model will oscillate if the resolution and time step are chosen to be too large. In general, the resolution should be chosen to be five to ten times smaller than the fastest current variation expected. In addition, the time step should be chosen to be two to five times smaller than the resolution time. If oscillations are observed, reduce the time step and resolution by a factor of two and try again. It is also wise to test any answer by running with a smaller time step to verify a correct solution.

An additional optional input parameter was also added. It is now possible also specify the electric field at which emission is turned on. If not specified, the turn on electric field is

200 kV/cm (2x107 volts/m). This input must be given after the resolution time, which is also optional.

A diagnostic specific to the racetrack MITL, the loss current density to the anode due to Child- Langmuir emission, has been added.

### Perveance MITL model

A turn-on electric field, identical to that used in the MITL model (although not yet variable), has also been added. To determine the electric field, the model must infer a transmission line gap from the perveance and line impedance. To do this the model assumes a coaxial transmission line and a constant gap. The model is working, but does not as closely reproduce the 2D particle-in-cell simulations results as does the racetrack MITL model. In general, it predicts more loss at lower currents and less loss at higher currents. However, the total energy loss is not significantly different than with the MITL model.

# Compiling Your Own Version of Screamer

The complete source code for SCREAMER is freely available and is considered OPEN SOURCE under the GNU License. All required subroutines to compile SCREAMER are contained in the SCREAMER package or are part of the Fortran Compilers that should be installed on your system. This chapter contains detailed instructions for compiling SCREAMER.

## Install the GNU Compiler Collection (GCC) and gFortran

Go to the GNU web site. (<http://gcc.gnu.org/)>The details of the GCC installation are provided. Follow the instructions carefully. Please note that there are multiple sites that have different installations of gcc. Proceed with caution.

You may choose to use a different Fortran compiler. Please feel free to do so. Some users have seen significant increases in performance when using the Intel compiler on Intel processors. The Intel compiler is available for purchase from Intel via the web.

### Example install

The GCC and gFortran collection for the Macintosh operating system are easily installed by going to the SourceForge High Performance Computing website ([http://hpc.sourceforge.net/index.php).](http://hpc.sourceforge.net/index.php)) Different versions of the GCC and gFortran compilers are available for different hardware and system variants of the Mac OS. The instructions are easy to follow. Installation should take less than 30 minutes.

## For the Macintosh – Install Xcode command line tools

If you are running on a Macintosh and would like to build your own version of SCREAMER you must install Xcode (free from the Apple Store) and you must install the Xcode command line tools. These tools are also available at no cost from the Apple Store.

## Copy the latest version of SCREAMER to your Computer

As of December 31, 2019, the latest version of SCREAMER is V4.4.1. This version is available from Idaho State University at iac.isu.edu/screamer.html. (Contact Rick B. Spielman [spierick@isu.edu](mailto:spierick@isu.edu) for details.) As the contacts change, modifications will be made to this manual. Please note that SCREAMER is constantly evolving. The latest version may not be V4.4.1.

Place the folder containing the SCREAMER sources in the location of your choosing. Make sure that you have placed it where you, the User, have write priviliges.

* 1. **Compiling and Linking SCREAMER**

It is assumed that the User will be building a version of SCREAMER in a LINUX shell (bash or equivalent). To do this open a terminal window that gives you access to the OS via LINUX or UNIX commands. Move to the Directory that contains the SCREAMER sources using the cd command. The SCREAMER sources contain an example build file. The file in Version 4.3.2 is called screamer64.bat. Note, as of V4.3, SCREAMER is a pure Fortran code with no c++ linkages. Even though SCREAMER is fully Fortran we are forced to use the following g++ build because gfortran has a bug that prevents the specification of the libquadmath.a static library. g++ does not have this bug. The file listing to build a fully static version of SCREAMER is:

gfortran -c -O03 -mcmodel=medium zdem.for \*.f ar crv screamer64.a \*.o

rm \*.o

ranlib screamer64.a

g++ -o screamer64 screamer64.a -static-libgcc /usr/local/lib/libgfortran.a

/usr/local/lib/libquadmath.a

cp screamer64 ../run\_decks/screamer64 rm screamer64.a

The build does the following steps:

1. Compiles all SCREAMER subroutines with the version of gFortran located in usr/local/lib. (In some LINUX installations gFortran make not be installed or is installed in a different location. Typing >gfortran -v will tell you if gFortran is installed.
2. Links the compiled Fortran with the necessary static libraries.
3. Copies a version of the SCREAMER executable to the directory containing the run decks.

You may choose to type the commands separately rather than execute the build through the csh or bash commands.

Note: the present 64-bit build of SCREAMER is nearly a fully static build. One can check to see if all of the libraries that are called with the final build are static or are always included in your OS by typing > otool -L screamer64. The final executable requires those dynamic libraries to be on any system from which SCREAMER is run. An example of such a call on macOS is given below.

$ otool -L screamer64 screamer64:

/usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1252.250.1)

$

This reponse shows a full static build with the only dynamic call being a standard Apple library.

## Running Your New Screamer Version

Once you have compiled SCREAMER you can run the new binary. We describe herein the details to run the sequential version Screamer from the Terminal Window NOT from the Finder or LINUX desktop. At the prompt simply type:

$ ./screamer64 runfile.txt

Where runfile.txt is simply the run file you would like Screamer to use. This command will execute SCREAMER. If the run deck is located in the local folder you simply type the name of the run file. If the run file is located elsewhere then you must type the complete path to it.

If you place a copy SCREAMER in /usr/local/bin/ with the cp command (It is likely you will have to type “sudo cp screamer64 /usr/local/bin/” and enter your password at the prompt.) Then simply typing “screamer64” anywhere will run SCREAMER. Remember to put the run deck in the same location as where you are running SCREAMER.

SCREAMER is designed to be run in large batch files. To do this simply create a batch.bat file and type all of the SCREAMER runs that are desired a line at a time. Each run must have its own runfile.txt associated with it. This is incredibly useful to run cases with a wide variation in parameters.

* 1. **SCREAMER Change Control**

One of the most confusing things Users can do is to make undocumented changes to the source code. While changes and improvements are highly encouraged as a means to improve the code, it is a problem if there are multiple, undocumented versions of the code available. Try to avoid this at all costs. SCREAMER is provided at no cost to all interested parties by Idaho State University at https[://www.isu.edu/iac/research/screamer/.](http://www.isu.edu/iac/research/screamer/) ISU will always have the latest “official” version of SCREAMER. These versions have been benchmarked against a large number of older run decks to make sure that errors have not crept into the code.

If a user must modify SCREAMER then we encourage two steps: first, document in detail the reason for the change and document the EXACT changes that were made to the sources (this includes all new subroutines and all changes to old subroutines) and, second, send an e-mail to ISU (spierick@isu.edu) with the details of the changes, the new and changed routines, and a request that these changes be part of the next formal SCREAMER release. Note: modifications to any subroutine MUST be shown in the modification log at the top of EACH subroutine AND in the master version changelog file.

If these changes are well documented, ISU can easily replicate those changes in future “official” releases, thereby maintaining a single, open version of SCREAMER that is up-to-date. If the changes are of general interest, they can be incorporated in the next official release of SCREAMER.

Remember, new releases of SCREAMER are tested against a wide range of run decks to look for bugs. Sometimes a change one place impacts the code somewhere else. Please provide run decks that test new parts of SCREAMER.

## Appendix Tips and Tricks

### A Modeling a Cylindrical MITL

The MITL model in SCREAMER uses a planar Child-Langmuir emission model, presuming that the transmission lines of interest are very low impedance and that the cathode and anode radii are very nearly the same. However, in a cylindrical geometry as larger impedances are used, the planar model over-predicts radial loss current. A factor that scales the area (or radius or circumference) in the planar formula can correct this problem.

The cylindrical form of the Langmuir current is given by Blodgett and Langmuir1. In *MKS* units the current is

*i* = *8**o*

*2e V3*  *2l*

*m*

*r* 

*2*

*a*

*9*

where *l* is the length of the coaxial transmission line, *V* is the voltage, *ra* is the anode radius, and ** is given as a series expansion.

 = *An**n n*

, where

 = *ln*  *ra*  *rc* .

Note that ** is just the line impedance divided by 60. The first few coefficients of *An* are

*n An*

0 0.0

1 1.0

2 -0.4

3 0.09166665

4 -0.01424242

5 0.001679275

This form of the Child-Langmuir law can be written in terms of an effective area *Aeff* similar to the planar form,

*i* = *4**o*

*2e V3*  *2A*

*eff*

*m*

*d2*

*9*

,

where *d* is the spacing between the electrodes and *Aeff* is

*e* – *1**2*

*Aeff* =

*Aplanarf*()

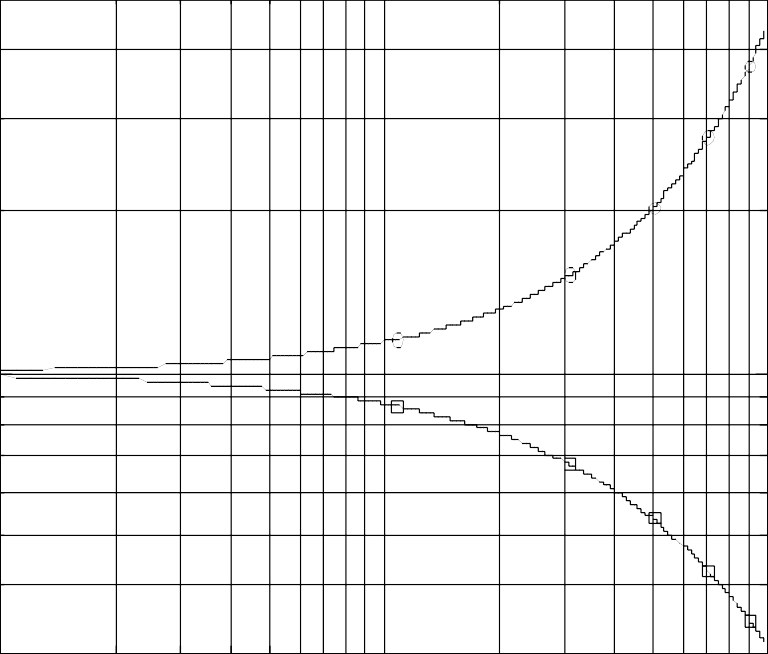
*f (*  =

*e**2*

and .

Thus, the correct cylindrical result can be obtained by using the planar model and scaling the cathode area, cathode circumference, or cathode radius by the factor *f()*. Note also that the factor depends on whether the cathode or anode radius is larger.

A plot of the correction factor for impedances between 1 and 100 Ohms is shown below. Below 1 Ohm the correction is not generally needed.

3

1

0.8

0.6

1 10 100





rc > ra

ra > rc

Fig. 16. Plot of the correction factor for impedances between 1 and 100 Ohms

1 K. B. Blodgett and I. Langmuir*, Phys. Rev*. **22**, 347 (1923).