



DeckBuild User's Manual

Silvaco, Inc.

2811 Mission College Boulevard
Santa Clara, CA 95054

Phone: (408) 567-1000

Web: www.silvaco.com

April 10, 2019

The information contained in this document is subject to change without notice.

Silvaco, Inc. MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE.

Silvaco, Inc. shall not be held liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

This document contains proprietary information, which is protected by copyright laws of the United States. All rights are reserved. No part of this document may be photocopied, reproduced, or translated into another language without the prior written consent of Silvaco Inc.

AccuCell, AccuCore, Athena, Athena 1D, Atlas, Blaze, C-Interpreter, Catalyst AD, Catalyst DA, Clarity RLC, Clever, Clever Interconnect, Custom IC CAD, DeckBuild, DevEdit, DevEdit 3D, Device 3D, DRC Assist, Elite, Exact, Expert, Expert C++, Expert 200, ExpertViews, Ferro, Gateway, Gateway 200, Giga, Giga 3D, Guardian, Guardian DRC, Guardian LVS, Guardian NET, Harmony, Hipex, Hipex C, Hipex NET, Hipex RC, HyperFault, Interconnect Modeling, IWorkBench, Laser, LED, LED 3D, Lisa, Luminous, Luminous 3D, Magnetic, Magnetic 3D, MaskViews, MC Etch & Depo, MC Device, MC Implant, Mercury, MixedMode, MixedMode XL, MultiCore, Noise, OLED, Optolith, Organic Display, Organic Solar, OTFT, Quantum, Quantum 3D, Quest, RealTime DRC, REM 2D, REM 3D, SEdit, SMovie, S-Pisces, SSuprem 3, SSuprem 4, SDDL, SFLM, SIPC, SiC, Silvaco, Silvaco Management Console, SMAN, Silvaco Relational Database, Silos, Simulation Standard, SmartSpice, SmartSpice 200, SmartSpice API, SmartSpice Debugger, SmartSpice Embedded, SmartSpice Interpreter, SmartSpice Optimizer, SmartSpice RadHard, SmartSpice Reliability, SmartSpice Rubberband, SmartSpice RF, SmartView, SolverLib, Spayn, SpiceServer, Spider, Stellar, TCAD Driven CAD, TCAD Omni, TCAD Omni Utility, TCAD & EDA Omni Utility, TFT, TFT 3D, Thermal 3D, TonyPlot, TonyPlot 3D, TurboLint, Universal Token, Universal Utility Token, Utmost III, Utmost III Bipolar, Utmost III Diode, Utmost III GaAs, Utmost III HBT, Utmost III JFET, Utmost III MOS, Utmost III MultiCore, Utmost III SOI, Utmost III TFT, Utmost III VBIC, Utmost IV, Utmost IV Acquisition Module, Utmost IV Model Check Module, Utmost IV Optimization Module, Utmost IV Script Module, VCSEL, Verilog-A, Victory, Victory Cell, Victory Device, Victory Device Single Event Effects, Victory Process, Victory Process Advanced Diffusion & Oxidation, Victory Process Monte Carlo Implant, Victory Process Physical Etch & Deposit, Victory Stress, Virtual Wafer Fab, VWF, VWF Automation Tools, VWF Interactive Tools, and Vyper are trademarks of Silvaco, Inc.

All other trademarks mentioned in this manual are the property of their respective owners.

Copyright © 1984 - 2018, Silvaco, Inc.

How to Read this Manual

Style Conventions		
Font Style/Convention	Description	Example
•	This represents a list of items or terms.	<ul style="list-style-type: none"> • Bullet A • Bullet B • Bullet C
1. 2. 3.	This represents a set of directions to perform an action.	<p>To open a door:</p> <ol style="list-style-type: none"> 1. Unlock the door by inserting the key into keyhole. 2. Turn key counter-clockwise. 3. Pull out the key from the keyhole. 4. Grab the doorknob and turn clockwise and pull.
→	This represents a sequence of menu options and GUI buttons to perform an action.	File → Open
Courier	This represents the commands, parameters, and variables syntax.	HAPPY BIRTHDAY
Times Roman Bold	This represents the menu options and buttons in the GUI.	File
<i>New Century Schoolbook Italics</i>	This represents the variables of equations.	$x + y = 1$
Note:	This represents the additional important information.	Note: Make sure you save often when working on a manual.

Table of Contents

Table of Contents	4
Chapter 1	
Introduction	8
1.1 What is DeckBuild	9
1.1.1 Features	9
Chapter 2	
Tutorial	12
2.1 Overview	13
2.2 Starting DeckBuild	14
2.3 Searching and Loading an Example	15
2.4 Running a Simulation	19
2.5 Plotting TonyPlot Files	21
2.5.1 Plotting Files from the Deck or Runtime Output	21
2.6 Quitting DeckBuild	22
Chapter 3	
Functions	23
3.1 DeckBuild Modes	24
3.2 Batch Mode Options	25
3.2.1 Examples	25
3.2.2 Preference Settings	26
3.3 Remote Mode	27
3.3.1 Introduction	27
3.3.2 Using Remote mode	27
3.3.3 Remote preferences	28
3.3.4 Prerequisites	34
3.4 DeckBuild Controls	35
3.4.1 The View Menu	35
3.5 Running Deck	38
3.6 Stop Points	43
3.7 History Feature	45
3.8 PDF report and movie generation	48
3.8.1 History Scripts	48
3.8.2 Movie creation	50
3.8.3 Browsing through history points	52
3.8.4 PDF report creation	53
3.9 Go to Line	54
3.10 Tracking Variables	55
3.11 Tracking Output Files	57
3.12 Tracking Resource Usage	59

3.13 Tools Menu	60
3.14 Edit Menu	61
3.15 Help Menu	62
3.16 File Menu	63
3.17 Examples	64
3.18 Cross-referencing runtime output and the deck	70
3.19 Folding runtime output	72
3.20 Visualizing VictoryProcess line statements	73
3.21 Context sensitive help system	75
3.22 Commands	81
3.22.1 Deck Writing Paradigm	81
3.22.2 Commands Menu	81
3.22.3 Parsing the Deck	81
3.22.4 Process Simulators	83
3.22.5 Writing a Process Input Deck	84
3.23 Preferences	88
3.24 Application	91
3.25 Tools	93
3.26 Editor Settings	94
3.27 History and File Settings	95
3.28 Runtime Settings	97
3.29 Simulation Settings	100
3.30 Registered Filetypes	102
3.31 Remote Settings	102

Chapter 4

Statements	103
4.1 Overview	104
4.1.1 DeckBuild Commands	104
4.2 ASSIGN	105
4.3 AUTOELECTRODE	109
4.4 DEFINE and UNDEFINE	110
4.5 EXTRACT	112
4.6 GO	113
4.7 IF, ELSE and IF.END	115
4.8 LOOP, L.END and L.MODIFY	116
4.9 MASK	118
4.10 MASKVIEWS	120
4.11 SET	121
4.12 SOURCE	124
4.13 STMT	126
4.14 SYSTEM	127
4.15 TONYPLOT	128

Chapter 5

Extract	129
5.1 Overview	130
5.2 Process Extraction	131
5.2.1 Entering a Process Extraction Statement	134

5.2.2 Extracting a Curve	136
5.3 Customized Extract Statements	138
5.3.1 Extract Syntax	138
5.3.2 DEFAULTS	182
5.3.3 Examples of Process Extraction	183
5.4 Device Extraction	192
5.4.1 The Curve	192
5.4.2 Curve Manipulation	194
5.4.3 BJT Example	196
5.5 General Curve Examples	197
5.5.1 Curve Creation	197
5.5.2 Min Operator with Curves	197
5.5.3 Max Operator with Curves	197
5.5.4 Ave Operator with Curves	197
5.5.5 X Value Intercept for Specified Y	197
5.5.6 Y Value Intercept for Specified X	198
5.5.7 Abs Operator with Axis	198
5.5.8 Min Operator with Axis Intercept	198
5.5.9 Max Operator with Axis Intercept	198
5.5.10 Second Intercept Occurrence	198
5.5.11 Gradient at Axis Intercept	198
5.5.12 Axis Manipulation with Constants	198
5.5.13 X Axis Interception of Line Created by Maxslope Operator	199
5.5.14 Y Axis Interception of Line Created by Minslope Operator	199
5.5.15 Axis Manipulation Combined with Max and Abs Operators	199
5.5.16 Axis Manipulation Combined with Y Value Intercept	199
5.5.17 Derivative	199
5.5.18 Data Format File Extract with X Limits	199
5.5.19 Impurity Transform against Depth	199
5.6 MOS Device Tests	201
5.7 Extracted Results	202
5.7.1 Units	202
5.8 Extract Features	203
5.8.1 Extract Name	203
5.8.2 Variable Substitution	203
5.8.3 Min and Max Cutoff Values	204
5.8.4 Multi-Line Extract Statements	204
5.8.5 Extraction and the Database (VWF)	204
5.9 QUICKBIP Bipolar Extract	205
5.10 Using Extract with Atlas	208
Chapter 6	
Optimizer	211
6.1 Overview	212
6.1.1 Features	212
6.1.2 Terminology	212
6.2 Using the Optimizer	213
6.2.1 Parameter settings	214
6.2.2 Target settings	214

6.2.3 Settings of the Optimizer	215
6.2.4 Running optimizations on curves	216
6.2.5 Optimizer return values	217
Appendix A	
Models and Algorithms	218
A.1 Introduction	219
A.1.1 Physical Models	219
A.2 Concentration Dependent Mobility	220
A.3 Field Dependent Mobility Model	221
A.4 Sheet Resistance Calculation	222
A.5 Threshold Voltage Calculation	223
A.5.1 Breakdown Voltage Calculation	224
Appendix B	
DBInternal	226
B.1 DBInternal	227
B.1.1 Example	227
B.2 The Template File	229
B.2.1 The trial_id Variable	229
B.3 The Experiment File	230
B.3.1 Load command	230
B.3.2 Experiment command	230
B.3.3 Save Command	230
B.4 Technical Details	231
B.5 DBInternal Commands	232
B.5.1 convert	232
B.5.2 doe	232
B.5.3 endsave	234
B.5.4 get_data	234
B.5.5 log	235
B.5.6 monte_carlo	236
B.5.7 no_exec	238
B.5.8 option	238
B.5.9 save	239
B.5.10 sweep	241
B.5.11 translate.ise	244
B.6 DBIT	245
B.6.1 The General Tab	246
B.6.2 The Matrix Tab	248
B.6.3 The Command Menu	250



Chapter 1

Introduction

1.1 What is DeckBuild

DeckBuild is an interactive, graphic runtime environment for developing process and device simulation input decks.

This is an extremely powerful and flexible tool that is easy to use and provides many automated features. It allows for transparent transition from one simulator to another, automatic definition of mesh and mask information, and application of built-in measurement (extraction) facilities. Before DeckBuild, these tasks often required user intervention and were extremely time consuming. By automating these tasks, DeckBuild allows you to concentrate on the real work at hand: accurate simulation.

1.1.1 Features

DeckBuild also offers several powerful features never before available. One of these features, the optimizer, allows optimization across an entire input deck even between different simulators. For example, varying an implant dose in SSuprem3 and a diffusion time in Athena permits optimizing against a Vt curve simulated with Atlas. DeckBuild also provides a seamless integration with DevEdit and its adaptive meshing capabilities. Also, the Utmost interface allows Silvaco's parameter extraction package Utmost III to load data from one of more device simulation runs to perform SPICE model parameter extraction. DeckBuild offers real flexibility with the ability to use UNIX system commands within simulation decks.

DeckBuild also contains many other convenience features:

- A built-in tool palette allows interactive plotting of the current structure.
- Full interactive control of the simulator, including a history function that allows you to back up in the deck and try again.
- The ability to define an arbitrary number of stop points where the simulator is halted automatically.
- An indication in the input deck of the currently executing line.

Simulators

Many simulators are available in DeckBuild and most are supported by a complete set of interactive popup windows. By selecting or moving various items on each popup, you can easily generate correct syntax. A deck is built by going through each desired popup and clicking on a **WRITE** button. This causes syntax to appear in the text editor. The deck can be saved and retrieved for later use. The popups have the additional feature of input-deck parsing. To do this, highlight a section of the input deck and choose **Parse Deck**. All appropriate popups will then re-configure themselves to reflect the syntax. For example, if you highlight an ATHENA IMPLANT statement and press **Parse Deck**, the Athena Implant popup will appear. This popup will reflect the values in the highlighted syntax.

For manual deck editing, DeckBuild has a built-in text editor with syntax highlighting. The text editor allows easy point-and-click editing, cut and paste to and from any other window, find/replace, multiple scroll views, and other features.

Auto-Interface

DeckBuild allows and encourages concatenating of decks from different simulators. For example, a simulation can start with SSuprem3 for fast 1D process simulation, move into Athena for 2D process simulation, and be followed by any number of separate Atlas

device tests. [Figure 1-1](#) shows a schematic of this flow. The entire result is saved as a single input deck.

Notice how process simulation is treated as a serial flow of events, while device simulations are treated as parallel. This is because of the way the auto-interfacing works. At the conclusion of each process run, the simulation results are saved, and are used by the next process simulator. Several process decks then form a serially-linked chain. Device tests always use the last available process result. Auto-interfacing is one of the most powerful features in DeckBuild.

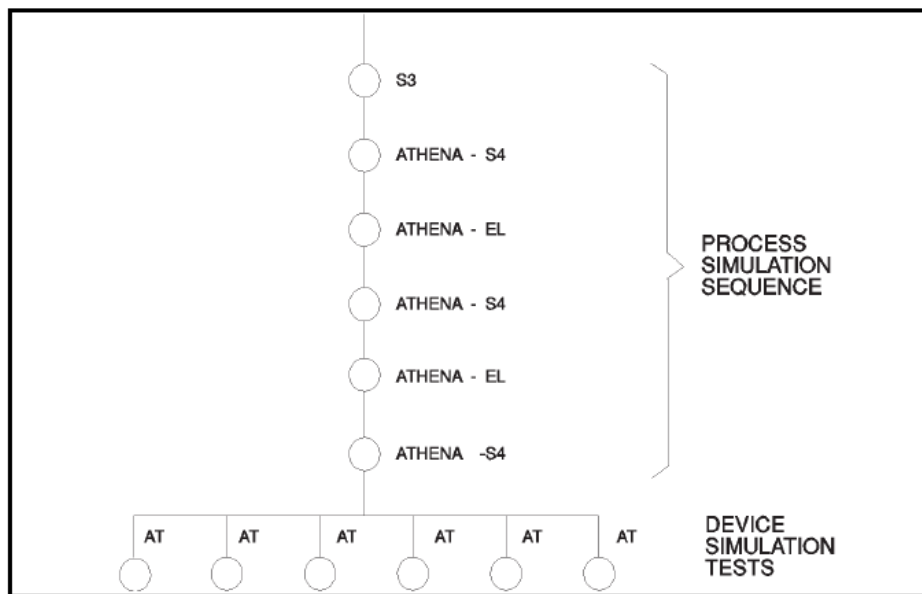


Figure 1-1 DeckBuild Flow

Execution Control

DeckBuild provides a diverse set of controls over the running simulation. You can run the entire deck. You can run it one line at a time. You can run the deck until a predefined line is reached or halted immediately after the current command. You can even set multiple stop points in the deck. While the simulation is running, DeckBuild also highlights the currently executing line in the input deck. The simulator itself can be stopped, quit, killed, paused, and unpaused.

One of DeckBuild's unique features is the **History** function (see [Section 3.7 History Feature](#)). DeckBuild remembers each line of the deck as it is executed and saves a structure file after each one. As a result, if a problem is discovered, it is unnecessary to redo the entire deck from the start. For example, if after running part way down an input deck and you discover a missing statement or an erroneous value, you only need to point and click on the line from which to start. DeckBuild automatically re-loads the saved history file and allows the simulation to continue from that point on.

DeckBuild also allows plotting structures created by the simulator in various ways. At any point in the deck, click a button and DeckBuild automatically causes the simulator to save a structure file. It then starts up Silvaco's post-processing tool (TonyPlot or TonyPlot 3D) using the saved structure as input. This is often useful in conjunction with **History** to aid in

fast tuning of a section of input deck. A statement can also be changed then re-executed, and the change is immediately visualized.

Examples and Tutorial

DeckBuild provides full on-line examples that can be loaded up at the press of a button. The examples are indexed so that you can enter search strings similar to search engines. These examples provide input decks for actual devices and help when learning about DeckBuild. [Chapter 2 Tutorial](#) is a tutorial that explains how to use DeckBuild to perform a simple simulation.

Advanced Uses

Generic Decks

Most decks have built-in geometric constants that reproduce a single, unchangeable cross-section of a wafer. DeckBuild's IC layout interface (MaskViews) makes it possible to write a single deck that can be used at any location on a wafer without using hard-coded geometry information. You can create (or read from GDSII or CIF format) device layout and mask layers using MaskViews. Then, create or modify a deck using DeckBuild to use mask names with hard-coded geometry values. Finally after making a cutline using MaskViews, DeckBuild can simulate that cross-section. You can simulate any cross-section of the device in this manner.

Extraction

DeckBuild contains built-in extract routines for both process and device parameter extraction. Extract forms a "function calculator" that allows you to combine and manipulate values or entire curves quickly and easily. You can take one of the standard expressions and modify it as appropriate to suit your needs or use the custom extract language to create unique extraction statements specific to the current simulation.

Extract also includes features, such as variable substitution and internal 1D device simulators, QUICKMOS and QUICKBIP for specialized cases of MOS and bipolar electrical measurement. All extracted results are displayed in the DeckBuild runtime output subwindow and stored in a datafile for easy comparison of different simulations.

Optimization

A powerful Optimizer is available within DeckBuild that allows quick and accurate tuning of simulation parameters. Specify any number of input parameters to vary and any number of targets to attain. For example, it is possible to find a target threshold voltage of 0.75 volts by varying gate oxidation time and V_t adjust implant dose.

Optimization parameters may come from any simulator supported by DeckBuild and targets from any extracted parameter. For example, it is easy to set up a deck that auto interfaces from SSuprem3 to Athena and then to Atlas. Then, extracted values can be optimized from I-V curves while using SSuprem3 or Athena diffusion coefficients or both as input parameters.



Chapter 2 Tutorial

2.1 Overview

In this tutorial a brief introduction to Deckbuild and its functionality are given. It will show you how to:

- Start DeckBuild
- Perform a basic search in the Examples Database and load an example
- Execute an example
- Plot the results
- Quit DeckBuild

If you are new to DeckBuild, please follow this tutorial guide. Once you are familiar with DeckBuild, you can see the remainder of the manual for details.

2.2 Starting DeckBuild

This section explains how to start DeckBuild and gives an introduction to the DeckBuild Graphical User Interface (GUI). To start DeckBuild in Linux, type the following into a terminal:

```
DeckBuild &
```

DeckBuild will launch and the window shown in [Figure 2-1](#) will appear. The upper portion of the window is the deck editor window, where you can either type in the syntax directly or display syntax from a loaded deck. The lower portion is the runtime output (RTO). This will display information generated from the simulation run. The very top is a range of drop down menus and toolbars.

The appearance of the editor can be adapted to your needs. Please see [Section 3.23 Preferences](#) for details.

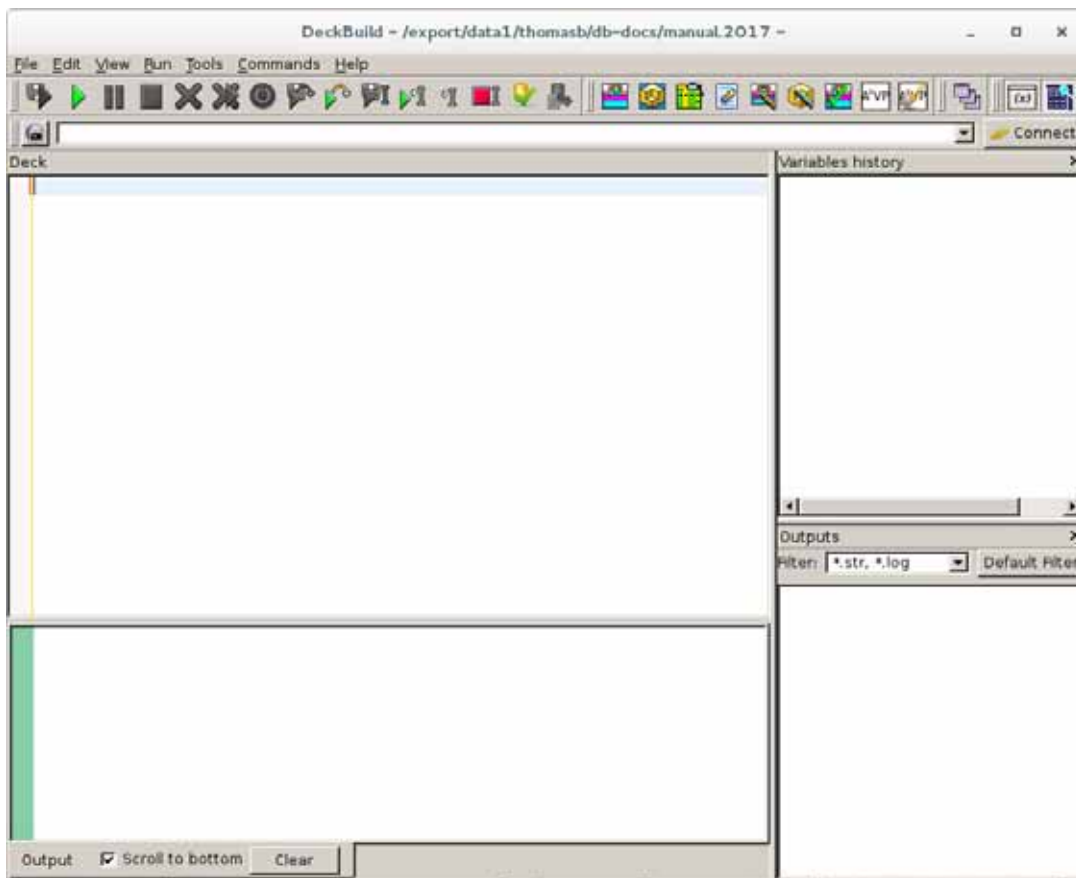


Figure 2-1 DeckBuild Graphical User Interface (GUI)

2.3 Searching and Loading an Example

In this tutorial, a standard example is going to be used to illustrate some of DeckBuild's features. More than 500 examples are shipped with DeckBuild. To access the examples, select **Files**→**Examples** as shown in [Figure 2-2](#).

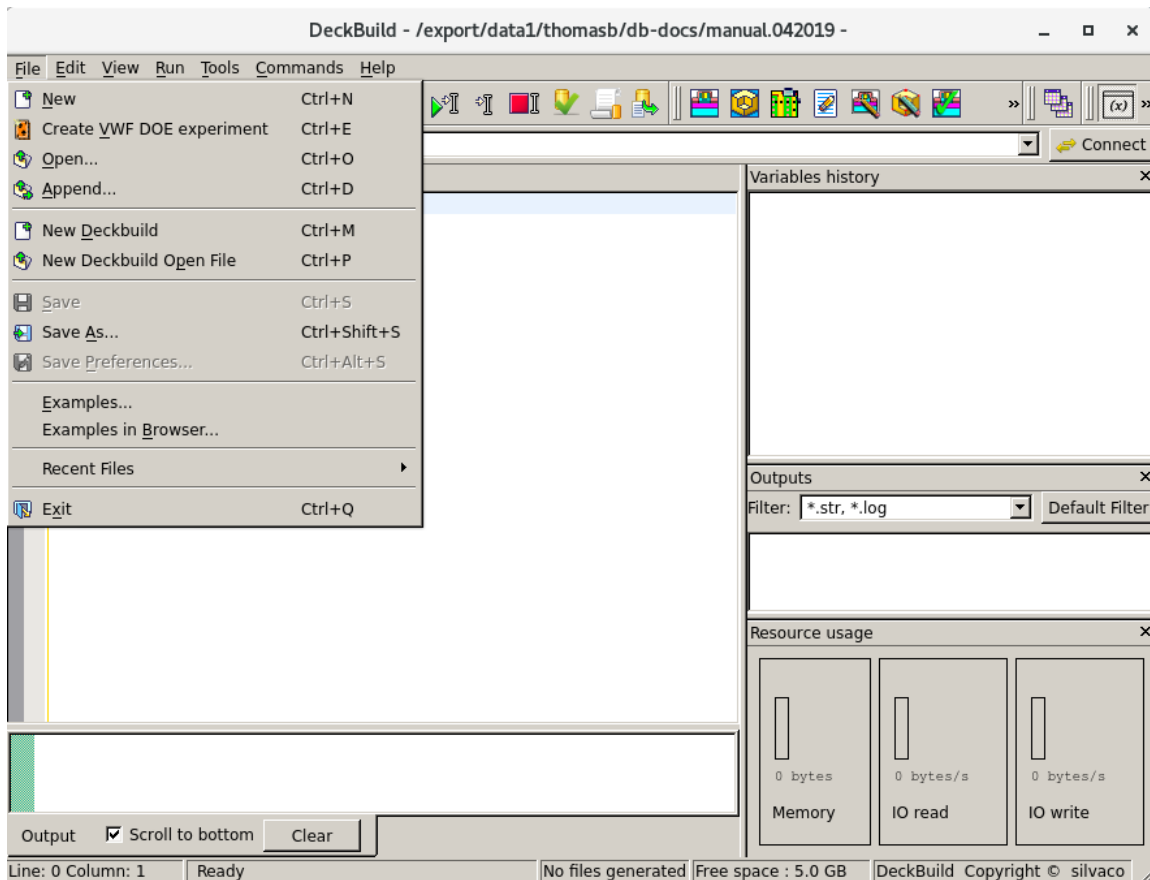


Figure 2-2 Opening the Examples Window

This will pop up the window shown in [Figure 2-3](#). The search dialog is initially populated with a hierarchical tree of examples.

By entering a search string, you can search the database of examples. By default, all fields are considered in the search. This includes the description, title, and section headings.

In this introduction, the standard example 'mos1ex01' will be used. To search for this example, enter the text 'mos1' into the search box. [Figure 2-4](#) shows the results. 'mos1' is also a section title and contained in the names of some other examples. Consequently, the search returns a number of hits.

Every example includes a description. This is displayed in the lower portion of the Examples search window.

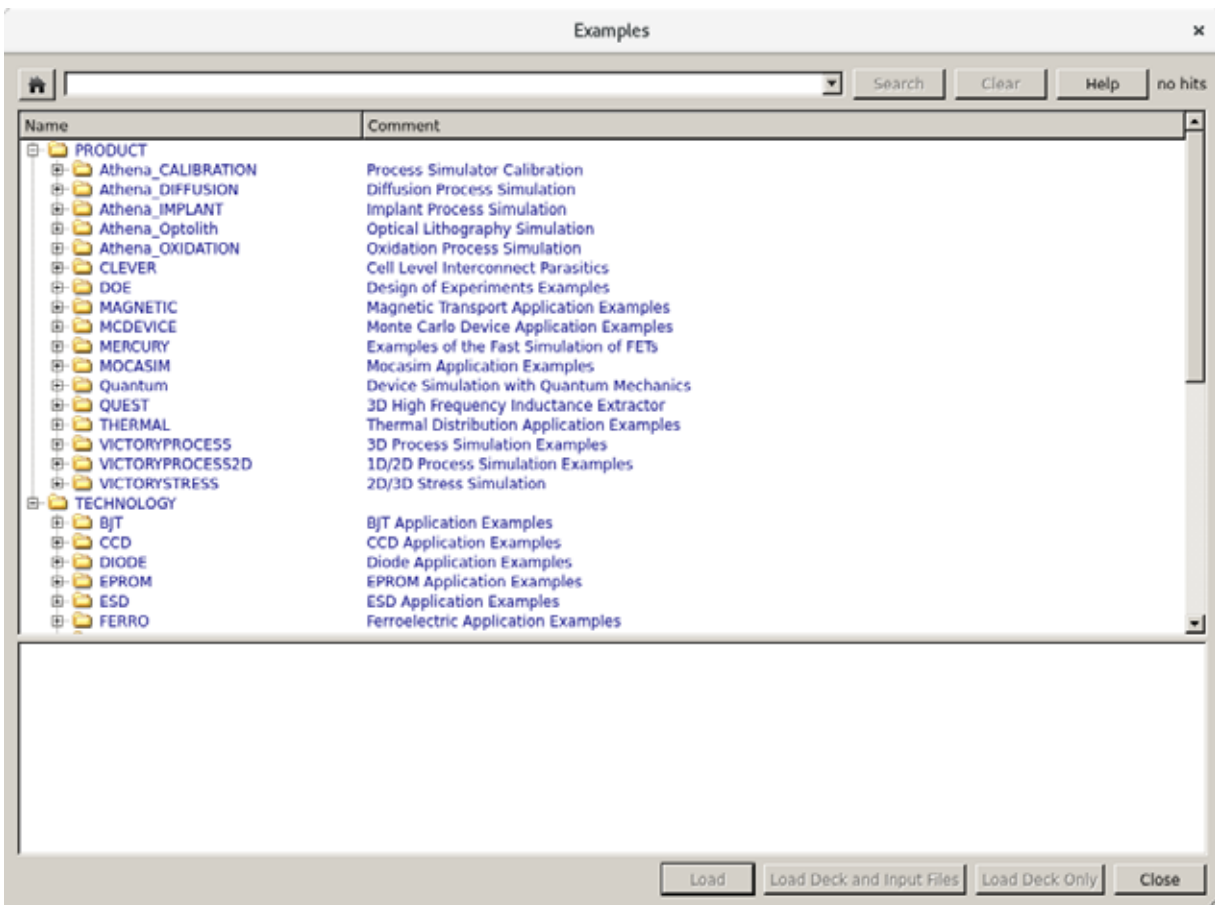


Figure 2-3 Examples Search Window

The screenshot shows the 'Examples' window in DeckBuild. At the top, there is a search bar containing 'mos1' and buttons for 'Search', 'Clear', and 'Help'. Below the search bar is a table with 16 hits. The first row is highlighted in blue. Below the table, a detailed view of the selected example is shown, including requirements and a cross-sectional diagram of a MOSFET.

section	example	title	simulator	technology	version
MOS1	mos1ex01	Id/Vgs and Threshold Voltage Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex02	Family of Id/Vds Curves	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex03	Sub-Threshold Slope Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex04	DIBL Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex05	Body Effect Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex06	Substrate and Gate Current Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex07	Breakdown Voltage Extraction	Atlas,Athena	NMOS	4.2.5.R
MOS1	mos1ex08	Id/Vgs and Threshold Voltage Extraction	Atlas,Athena	PMOS	4.2.5.R

Id/Vgs and Threshold Voltage Extraction

Requires: SSuprem 4/S-Pisces
 Minimum Versions: Athena 5.22.3.R, Atlas 5.26.1.R

ATHENA
 Data from mos1ex01_0.atr

The diagram shows a cross-section of a MOSFET with a red gate stack, blue channel, and yellow/red source and drain regions. The y-axis is labeled 'source' and 'drain' with values from 0 to 0.4.

Buttons at the bottom: Clear history, Load, Load Deck and Input Files, Load Deck Only, Close.

Figure 2-4 Search Results for String 'mos1'

You can refine your search by using additional control parameters, such as adding more keywords, using commands to exclude keywords, and limit the fields searched. Further details can be found in [Section 3.2.1 Examples](#).

Example 'mos1ex01' is highlighted in [Figure 2-4](#). Clicking on the **Load** button will load it into DeckBuild.

Clicking on another example will highlight it. The description in the lower window will also change to reflect the new selection. You can also double-click on an example to load it.

[Figure 2-5](#) shows the deck loaded into DeckBuild.

All standard examples are supplied already executed. Therefore, when you load a standard example, all of the files for that example are placed into the current working directory. This means that you do not have to execute a standard example to inspect the results.

Copying the result files can consume both time and memory. The dialog shown in [Figure 2-4](#) offers a way to load only the simulation deck without copying all the other files. To do this, click the **Load deck only** button. To load a pre-existing deck rather than an example, select **File→Open** from the drop down menus at the top of the window.

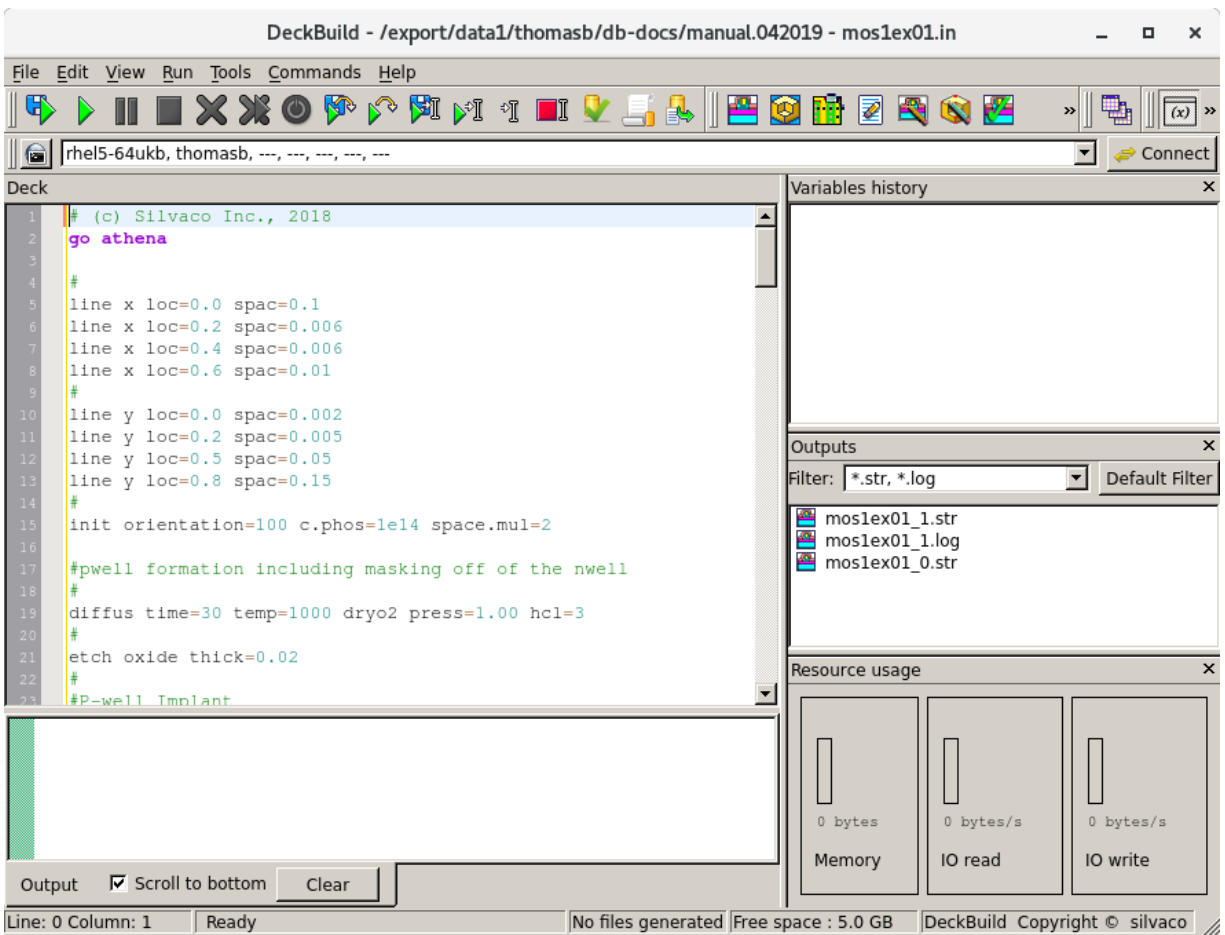


Figure 2-5 DeckBuild with mos1ex01 Loaded

2.4 Running a Simulation

DeckBuild is incredibly flexible on how to run a deck. You can run line-by-line, run up to a certain point, jump forward and backwards or any combination thereof. You can also run a deck from beginning to end.

Figure 2-6 shows the options available when you click on **Run**. The very same functions are also provided by default on the Toolbar.

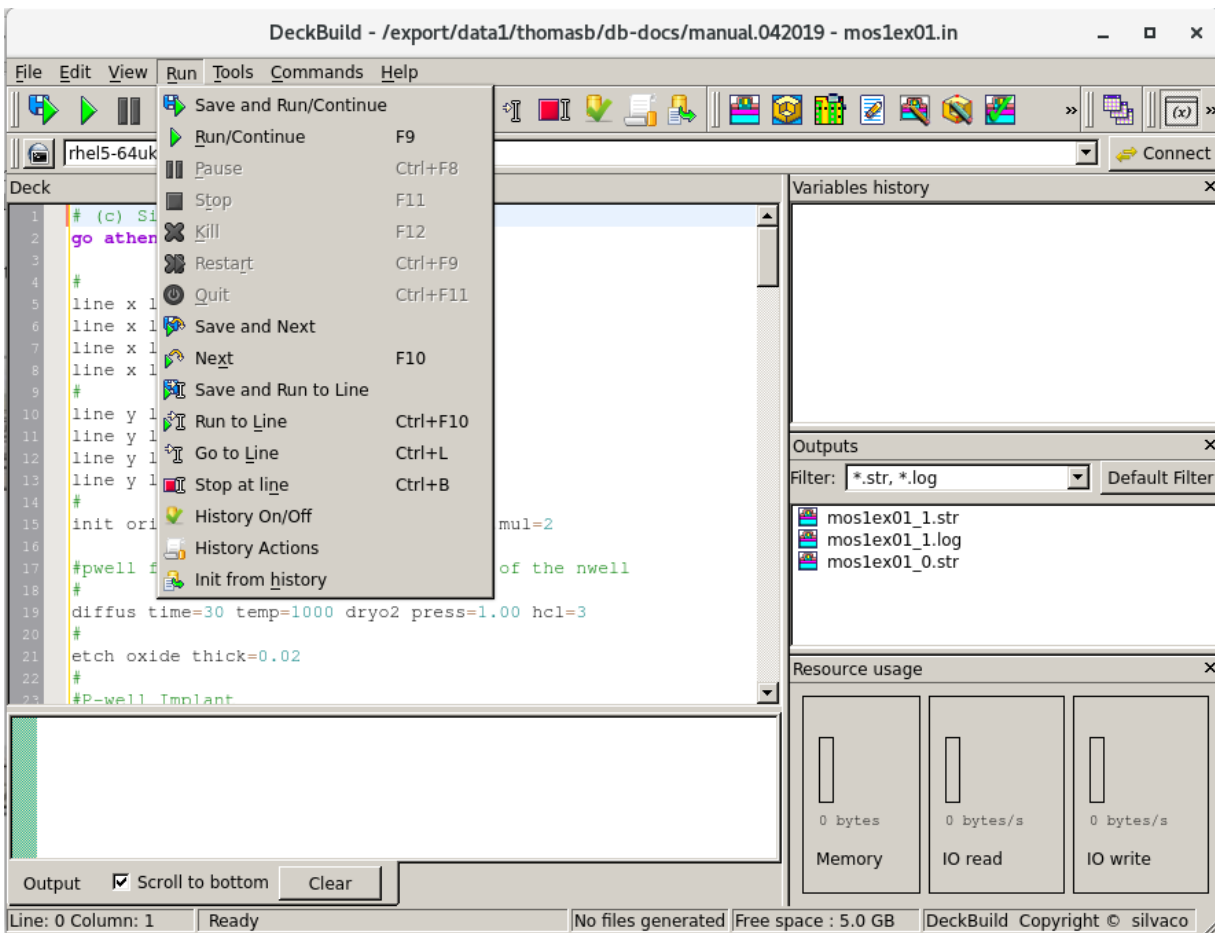


Figure 2-6 Options to Run Deck as Available from the Run Menu

To run the deck through from top to bottom click on the **Run** button on the toolbar (▶) or drop down menu.

The deck will immediately be executed and the RTO will be populated with messages from the simulator (see Figure 2-7).

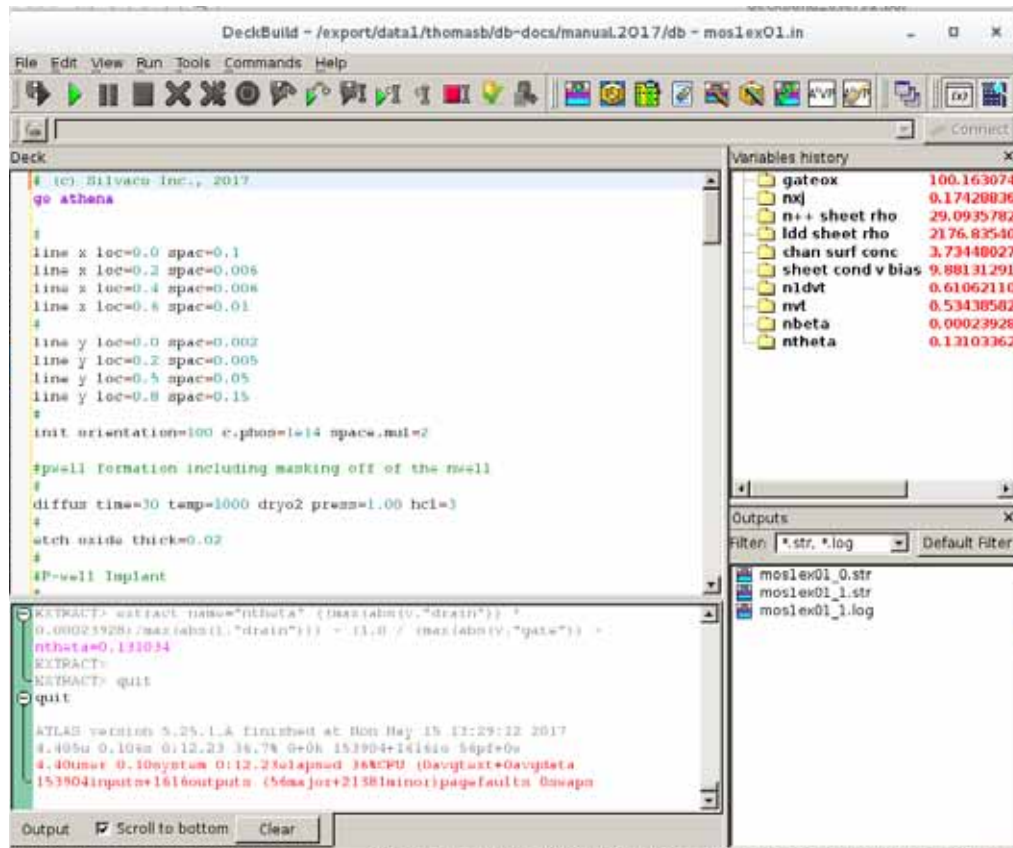


Figure 2-7 Running Deck

In certain simulators, history files are generated. DeckBuild automatically inserts `SAVE` statements into the deck at regular occurrences to generate these files. A history file preserves the state of the simulation, so you can later re-run a simulation from any saved point. This is a great speed up since you only need to edit and re-run portions of the deck rather than re-executing the entire simulation from the top again after deck was edited.

The presence of a history file is indicated by the yellow dot next to the statement in the editor window (see [Figure 2-7](#)). The accompanying `SAVE` statement is shown in the runtime output pane (`struct outfile="...../history.."`).

For more information on re-initializing a simulation using a history file, see [Section 3.7 History Feature](#).


Information on alternative methods of running decks, such as line-by-line and breakpoints can be found in [Section 3.5 Running Deck](#).

2.5 Plotting TonyPlot Files

When a deck is executed, it generates a number of files, such as structure and log files (.str and .log). These files are displayed using TonyPlot.

2.5.1 Plotting Files from the Deck or Runtime Output

Wherever a TonyPlot recognizable filename (.str and .log) appears in the deck or the RTO, you can right-click the filename to plot the file. Figure 2-8 shows the menu that opens when you right-click on a valid filename. Clicking on **Plot ...** will cause TonyPlot to launch, displaying the specified file.

You can also launch TonyPlot from DeckBuild by clicking on the **TonyPlot** icon () on the Toolbar or by selecting **Tools**→**TonyPlot**.

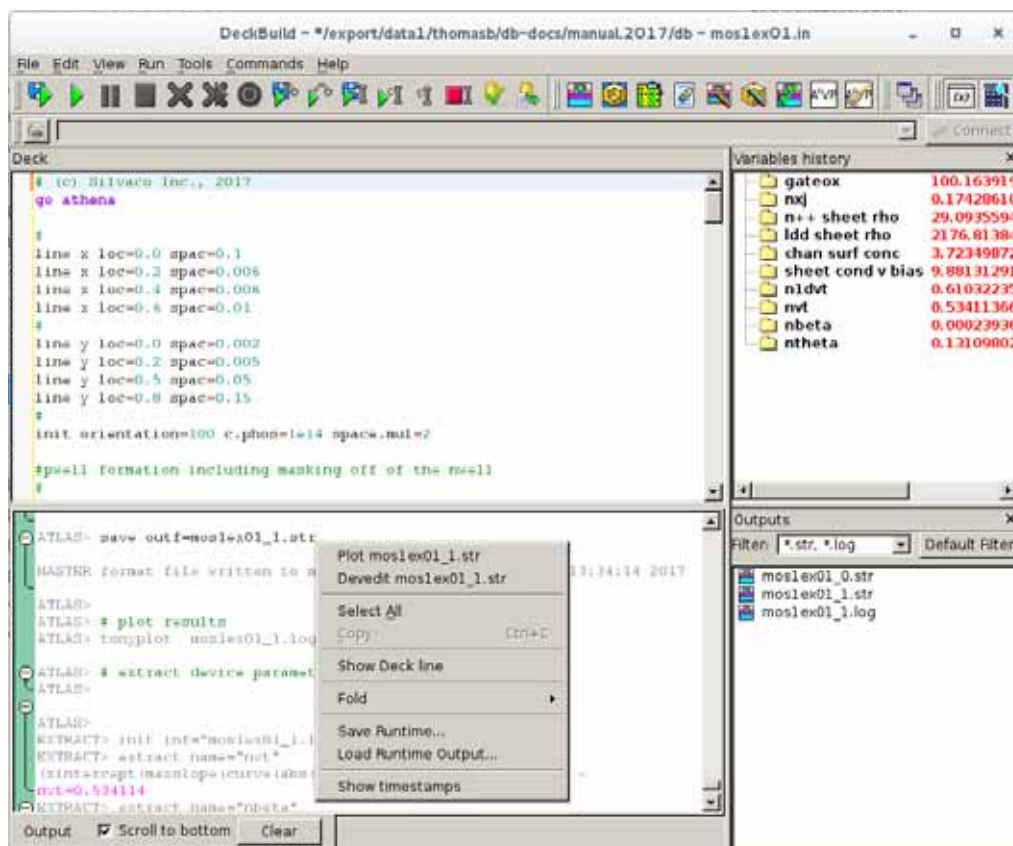


Figure 2-8 Plotting Files Specified from the Runtime Output

2.6 Quitting DeckBuild

To quit DeckBuild, select **File**→**Exit** or click on the X in the top right-hand corner.



A certain number of files are automatically generated during a simulation run (e.g., history, runtime, and output logs). These files are saved in a hidden folder in your current working directory. After the example 'moslex01' has been run, DeckBuild will ask you to confirm if these files will be kept or removed (Figure 2-9).

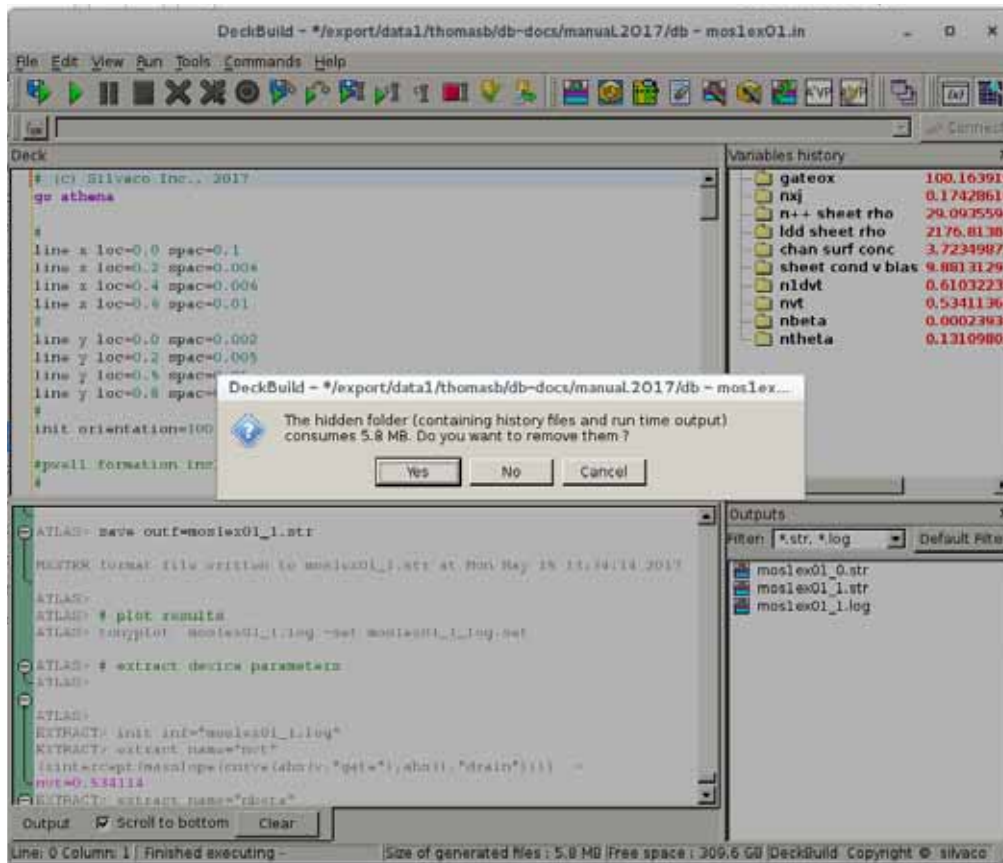


Figure 2-9 Terminating DeckBuild – Remove Generated File



Chapter 3

Functions

3.1 DeckBuild Modes

You can start DeckBuild in either an interactive mode or a batch mode. In the interactive mode, you can create, edit, and run input decks using mouse and keyboard operations. In the batch mode, DeckBuild runs a previously created input deck. In the interactive mode, DeckBuild appears as a window containing GUI components. The filename specifies the file to edit. If not specified, the editor pane will be empty. If specified, DeckBuild loads the file into the editor pane.

In batch mode, a filename is required. DeckBuild automatically starts the simulation and executes the entire input deck. DeckBuild quits when the run is complete. In batch mode, you can save the run-time output of the simulation by specifying the `-outfile` option.

There is another mode that DeckBuild can be run. This is to update the examples index, which is used in the examples search dialog. You need to use this mode everytime you install a Silvaco Software packages over a previously existing install location. Typically, this is the case when you install an update package. To invoke this mode the option `-rebuild_index` has to be used in the following way:

```
thomasb@lannachn2$ deckbuild -rebuild_index /site/alpha/examples/deckbuild/4.2.1.R
.....
.....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_adaptmesh.....
indexing                               section:/site/alpha/examples/deckbuild/4.2.1.R/
athena_advanced_diffusion.....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_calibration.....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_complex.....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_compound....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_diffusion.....
indexing                               section:/site/alpha/examples/deckbuild/4.2.1.R/
athena_elite.....
indexing                               section:/site/alpha/examples/deckbuild/4.2.1.R/
athena_implant.....
indexing section:/site/alpha/examples/deckbuild/4.2.1.R/athena_misc.....
.
.
.
renaming previous index /site/alpha/examples/deckbuild/4.2.1.R/lucene_index --> /site/alpha/
examples/deckbuild/4.2.1.R/lucene_index_old
renaming previous /site/alpha/examples/deckbuild/4.2.1.R/db-examples.xml --> /site/alpha/examples/
deckbuild/4.2.1.R/db-examples_old.xml
renaming /site/alpha/examples/deckbuild/4.2.1.R/lucene_index_new --> /site/alpha/examples/
deckbuild/4.2.1.R/lucene_index
saving /site/alpha/examples/deckbuild/4.2.1.R/db-examples.xml
```

In case an index is already existing deckbuild will ask you to manually remove it first and will exit. You will get a message as below:

```
thomasb@lannachn2$ deckbuild -rebuild_index /site/alpha/examples/deckbuild/4.2.1.R
directory "/site/alpha/examples/deckbuild/4.2.1.R/lucene_index_old" contains a previously created
index and needs to be removed manually
```


3.2 Batch Mode Options

The following options are used to run decks in batch mode:

- **-run** starts DeckBuild in batch mode. The input deck filename is required. If none is specified, DeckBuild displays an error message and exits.
- **-outfile <outfile>** is the run-time output file. The file specified by `outfile` is created to store the run-time output of the simulation. DeckBuild writes each line of the tty subwindow to `outfile` as the simulation progresses.
- **-ascii** enables DeckBuild to run in a non X windows environment. No popups or windows are created, but an input deck can be run normally. This option requires the use of an input filename and the `-run` option. The `-outfile` option (to store run-time output) is also helpful. If `-outfile` is not specified, the run-time output goes to `stdout`. For backwards compatibility reasons (VWF), you have to use both `-run` and `-ascii` at the same time to run a job in batch mode.
- **-opt <optimizerfile>** starts the deckbuild optimizer in batch mode. Please refer to [Chapter 6 Optimizer](#) for further details on how to run an optimization.
- **-err[file] <stderr-file>** redirects all `stderr` to the given file
- **-out[file] <stdout-file>** redirects all `stdout` to the given file
- **-nice nice-increment** sets nice value to the given number. Valid numbers are 1 to 19. Negative numbers can not be given.
- **-preferences <preferences-file>** loads certain preferences settings from the given file. Settings that are read are the auto-interface settings as well as the selected installation path and version of a simulator. Any other settings stored in the preferences file are ignored.
- **-help** displays a list of the DeckBuild command line options.

3.2.1 Examples

The following command will start DeckBuild in interactive mode and pre-load the specified file.

```
deckbuild filename.in &
```

DeckBuild can be submitted as a batch command on the UNIX command line. This method runs an input deck and quits at the end of the deck. You can submit a number of jobs for serial execution in this manner. The format of the command uses the `-run` and `-ascii` options as follows:

```
deckbuild -run -ascii filename.in
```

DeckBuild immediately starts executing the named input deck. DeckBuild exits completely when the last command in the input deck has been executed. If the runtime output is required to be stored into a separate file, the following options can be used:

```
deckbuild -run -ascii [filename.in] -outfile [filename]
```

Again, DeckBuild executes the specified input deck. But in this case, all runtime output is appended to the named `outfile`.

Be careful when saving structure files during batch jobs. Make sure to avoid overwriting structure files with subsequent runs in the same working directory.

3.2.2 Preference Settings

A large number of control settings and options are configured via **Preferences**. To configure these options, select **File**→**Preferences**.

Note: DeckBuild automatically changes simulators whenever it encounters an `autointerface` statement in the input deck.

3.3 Remote Mode

3.3.1 Introduction

Deckbuild supports a mode of operation whereby the simulation is carried out on a remote server machine. This allows you to separate the simulation action from the graphical user interface. This becomes useful when you are running your simulations on a client/server environment. Typically, you want to run the simulation on a powerful server machine (CPU, RAM) while the visualization takes place solely on your workstation. This is particularly important in case you need 3D graphics acceleration to display large 3D structure files. The TonyPlot v5 visualization tool utilizes the OpenGL-2.0 subsystem, which will not generally be available on remote server environments.

3.3.2 Using Remote mode

To use the remote mode of Deckbuild you need to first configure the machine to connect to as well as a directory on the server where all the simulation files are created. We will first give an example on how to connect and then point out the various dialogs and settings that are needed to define the server setup.

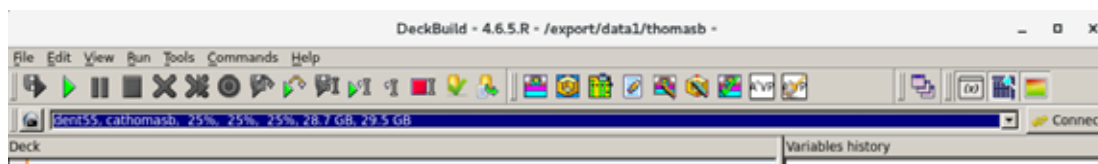


Figure 3-1 GUI widgets to control remote mode

Figure 3-1 displays the components to control the remote mode. The controls consist of a button showing a remote shell symbol (⌨), a text/pull-down area and a **'Connect'** button. By clicking on the **'Connect'** button, deckbuild will login (via the SSH protocol) to the system indicated in the text area using the displayed username (in this example the machine is called dent55 and the username is cathomasb.) While the connection is established a progress dialog is shown to indicate the status. Figure 3-1 also shows load and memory information of the remote host. The first three numbers indicate the 1min, 5min and 15min load averages (same as what you get by the top or upload commands on Linux). The remaining two numbers indicate available and total amount of memory.

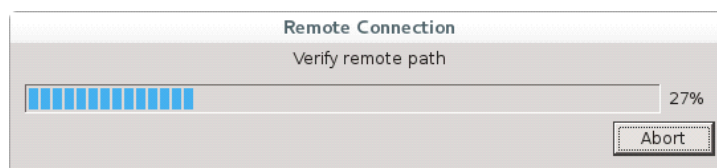


Figure 3-2 Progress

Figure 3-2 shows a screen-shot of the progress dialog. During the connection phase the dialog will indicate various stages of the connection process and will then disappear as soon the connection was established successfully. In case you want to interrupt the connection phase you can click on the **'Abort'** button, which will close the dialog and immediately abort the connection process. During the connection phase the system establishes one SSH, one or several SFTP and a CORBA connection.

When the connection was established, the looks of the remote mode components will slightly change to indicate that.



Figure 3-3 Successful server connection

Figure 3-3 shows that the text-area has been grayed out and the 'Connect' button now changed to read 'Disconnect'. The connection has been established successfully.

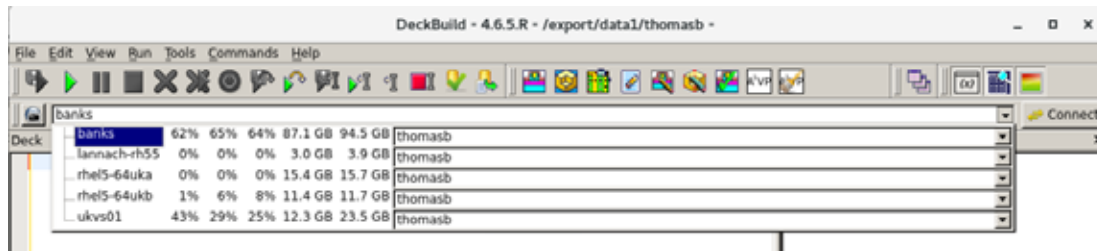


Figure 3-4 Hosts and usernames available for connection

The area, which displays the host and username can actually be used to display all available host names, usernames as well as load and memory information. The information is shown when you click on the arrow at the right. This is displayed in Figure 3-4, which shows a total of five machines: banks, lannach-rh55, rhel5-64uka, rhel5-64ukb and ukvs01 as well as their respective load and memory information. The expanded drop down menu will also allow you to chose a different username, should you have configured several for a given machine.

3.3.3 Remote preferences

Deckbuild will store connection details like a list of servers and usernames, the authentication mode (password or public/private key) as well as other related settings in its preferences.

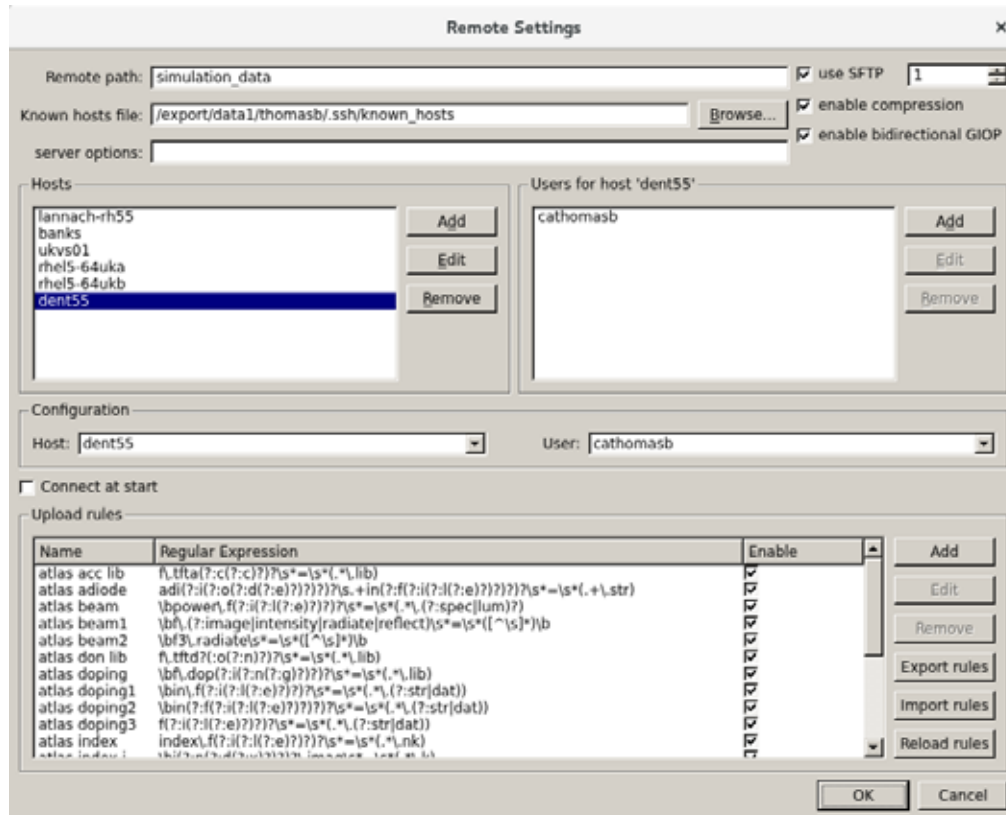


Figure 3-5 Preferences settings for remote mode

Figure 3-5 displays the settings panel for the remote settings. You can open this dialog by clicking on the remote shell symbol from the main window or – alternatively – by opening the preferences panel via **Edit** → **Preferences** as shown in Figure 3-6. The content of the panel is the same no matter how you opened it.

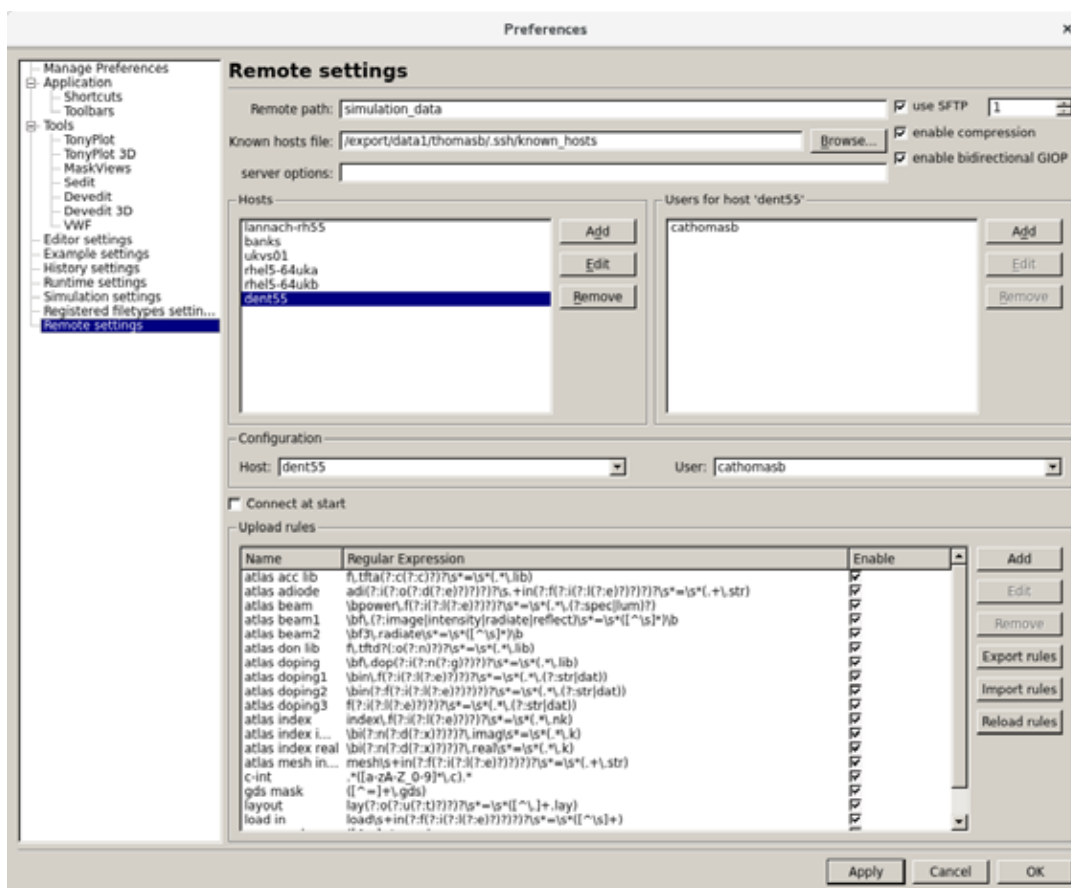


Figure 3-6 Preferences settings

The top part of the remote preferences consists of the following settings:

- **Remote path** – This is the path that is used on the server side to run the deck. The directory must exist on the server. Please note, that this setting is not related to the client directory structure in any way.
- **Known hosts file** – The SSH subsystem contains a so-called “known hosts” file, which is a list of all hosts that were visited (logged in) before. This file is shared with the SSH system installed on your computer. Therefore, any host that you already logged in via SSH on the command line is also known to the deckbuild remote mode.
- **server options** – This field allows you to pass extra options down to the server process that is started on the remote side. This is useful only for diagnosing problems and is not needed during regular operation.
- **use SFTP** – Allows you to use SFTP for downloading and uploading files. The only reason why you may want to disable SFTP is when you are using NFS to share folders/files between client and server. You must be using the exact same directory name space on both client and server in this case. Files are not uploaded/downloaded then but are directly taken from the filesystem. This will not work on Windows clients unless you are using a Windows NFS client and run deckbuild from a cygwin shell. It is strongly recommended to always use SFTP when your client is a windows system.

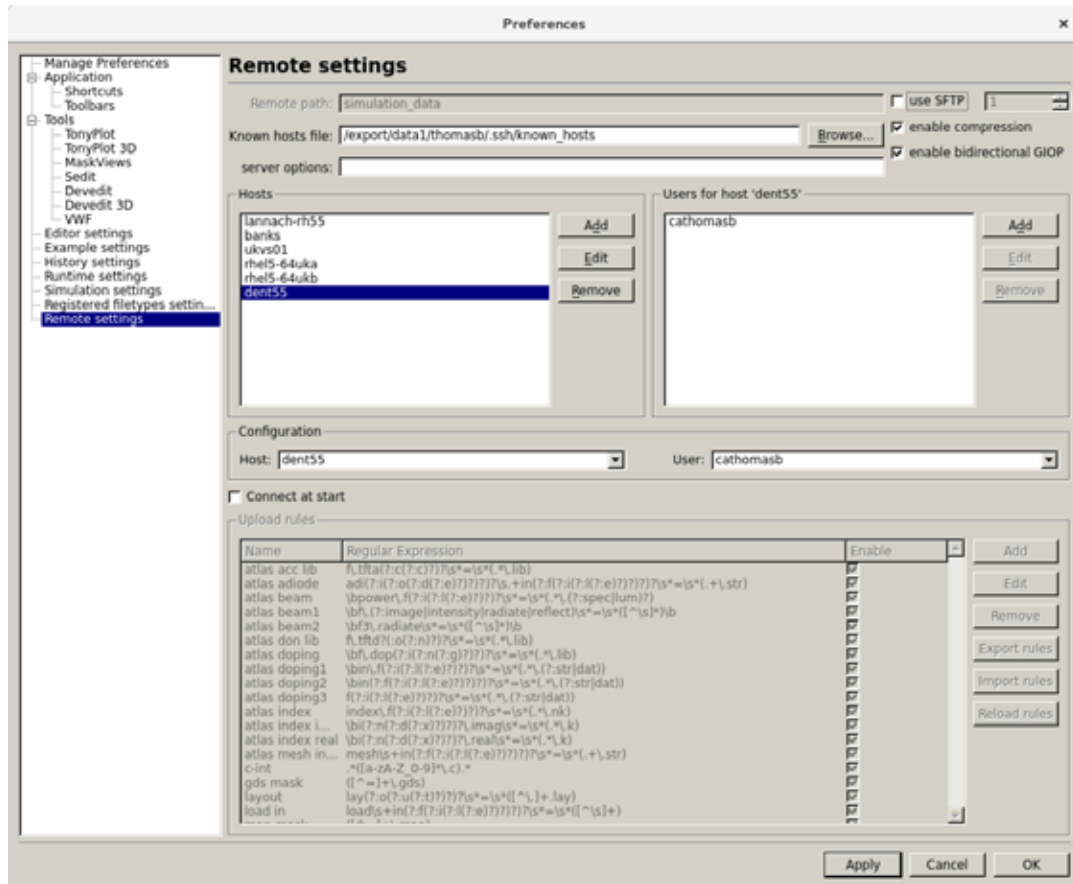


Figure 3-7 use SFTP disabled

Note, that, when 'use SFTP' is unticked, some of the remote settings do not make sense and are thus greyed. Particularly, the upload rules (mentioned below) are not used without SFTP. See [Figure 3-7](#).

The field next to the tick box allows you to define a maximum number of SFTP connections that are utilized in parallel. It is a good idea to start with a value of 1 and only increase in case you experience performance problems during up- or download.

Files normally being downloaded are structure files (.STR) or .LOG files that are typically created during the simulation run. In case the simulation requires an input file (e.g. a mask) then these input files are uploaded to the server and at the time right before the deck line requiring the input file is executed.

- **enable compression** – Will compress all CORBA network traffic between client and server. It is recommended to be set to true.
- **enable bidirectional GIOP** – Will use a single CORBA connection for in- as well as outgoing data. This is helpful in case there is a firewall between client and server. It is recommended to be set to true.

The middle part of the remote preferences consists of the login details that are used. You can keep an arbitrary number of hosts and username combinations. Usernames are stored on a per host base, you need to redefine every username for every host that you want to use it on. Clicking on the 'Add' button will open the dialog shown in [Figure 3-8](#).

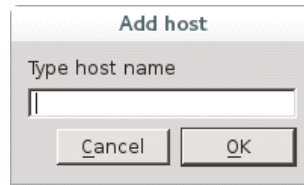


Figure 3-8 Adding a host

Clicking on **Edit** will allow you to modify a previously added host as shown in [Figure 3-9](#).

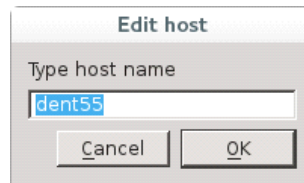


Figure 3-9 Editing a hostname

The **Remove** button will allow you to remove a hostname. If you remove a host then all corresponding usernames are removed as well.

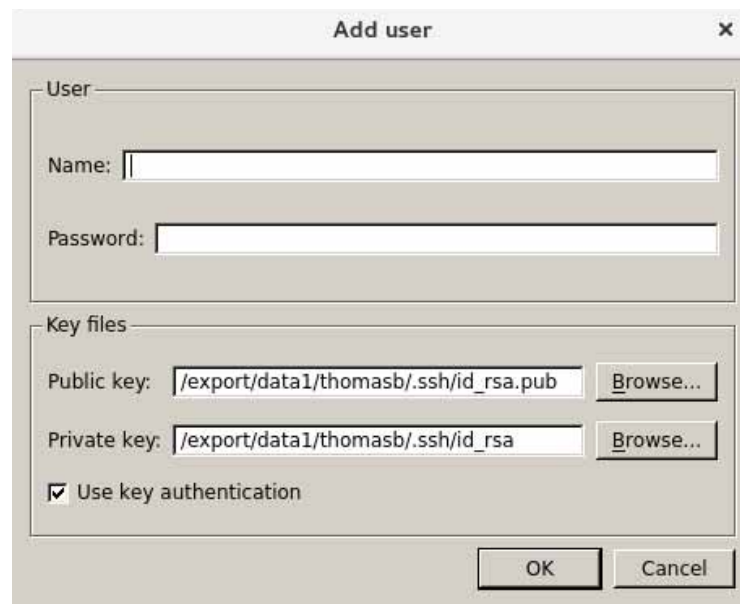


Figure 3-10 Adding usernames for a host

[Figure 3-10](#) depicts the dialog to add a new user for a selected host. You must enter a name and optionally a password or the public and private keys when public/private key authentication is to be used. When you enter a password this will be stored encrypted in your preferences. To use public/private key authentication the checkbox ‘Use key authentication’ must be ticked.

[Figure 3-11](#) shows the dialog, which opens when you click on the **Edit** button. It allows you to change either the username, password or the keys that are defined for a particular user in case public/private key authentication is used.

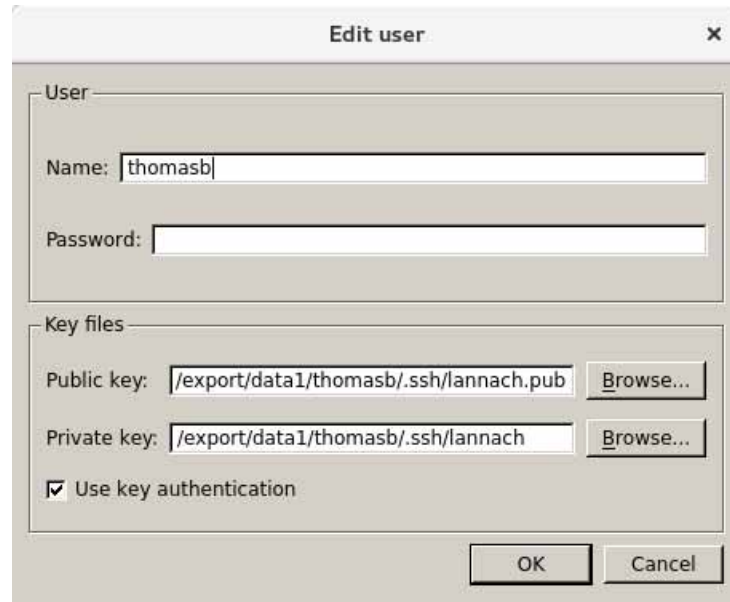


Figure 3-11 Editing a username

Right below the list of host and usernames (c.f. [Figure 3-5](#) and [Figure 3-6](#)) you can select, which combination of host and username is being used for the connection (when clicking 'Connect' in the main window, c.f. [Figure 3-1](#)).

By ticking the 'Connect at start' tick box you can have deckbuild login to the server automatically every time it starts.



Figure 3-12 Upload rules

[Figure 3-12](#) shows the final part of the remote preferences located at the very bottom of the panel (see also [Figure 3-5](#) and [Figure 3-6](#)). It contains a list of regular expressions. These expressions are applied to every line of the deck to identify any possible input files. If an expression matches a given line of deck then the identified input file is uploaded before the deck line is executed on the server. This ensures that all needed input files are found on the server.

Please note, that deckbuild ships with the above listed rules as a default setting. Should

your set of rules look different it is a good idea to either reset your preferences to factory defaults or to click the 'Reload rules' button in order to get the default list of rules. Please note the 'Import rules' and 'Export rules' functionality, which can be used to export rules to a text file and import rules from a text file respectively. This is useful to allow for an update of upload rules without having to upgrade to a newer version of deckbuild..

3.3.4 Prerequisites

In order for the remote mode of Deckbuild to function proper, you need to make sure that no firewall (hard- or software) is operating between the client and the server machine. It is not sufficient to only support the SSH protocol as the remote mode uses the CORBA middle ware to communicate between client and server.

3.4 DeckBuild Controls

DeckBuild consists of a window containing two subwindows: the Editor pane in the upper half of the base frame and the Runtime Output pane in the lower half. The Editor pane is used to build and edit input decks, while the Runtime Output pane is used to display feedback from a running simulator.

DeckBuild's controls are available in the toolbar and menus (see [Figure 3-20](#)). The following describes these controls.

The following terms and definitions will be used throughout this document to describe one of several states a simulation may be in.

- A simulator will be called `executing` if it is running a line of deck.
- A simulator will be called `waiting` if it presents a prompt, waiting for user input. It will achieve this status as soon as a `deck` statement is finished and no further statements are to be executed (e.g., after single stepping).
- A simulator will be called `paused` if it is possible to later continue the simulation exactly at the point where it was paused by using `unpause`.
- A simulator will be called `terminated` if it has ended either forcibly or gracefully (by means of **Quit**).

3.4.1 The View Menu

The **View** menu is used to change the appearance of the editor and the Toolbars (see [Figure 3-13](#)).

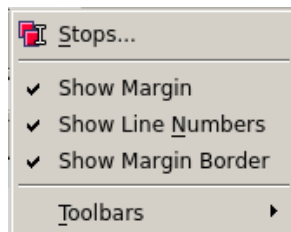
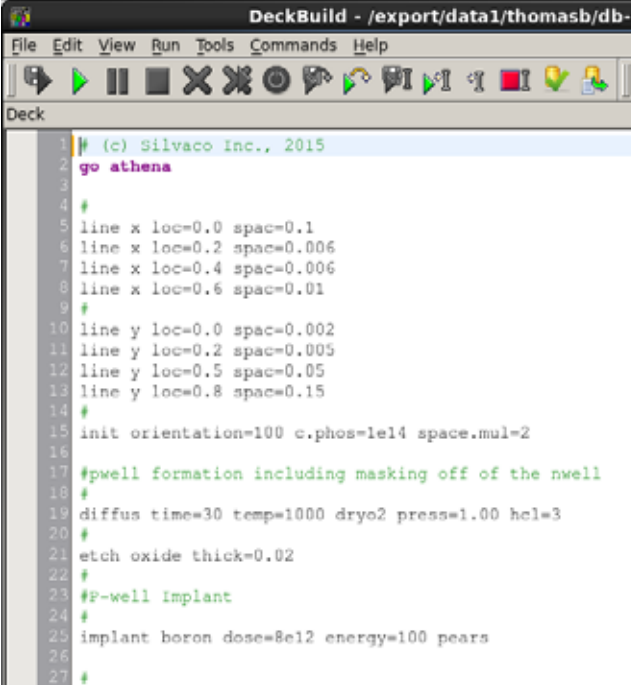


Figure 3-13 View Menu

This allows you to define what is displayed in the left margin of the editor, what toolbars will be available, and to open the Stops dialog (see [Figures 3-14](#) and [3-15](#)).



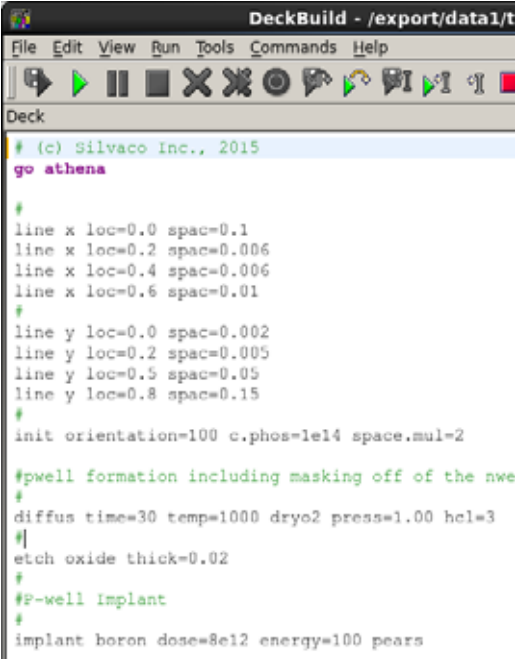
The screenshot shows the DeckBuild application window with the title bar "/export/data1/thomasb/db-d". The menu bar includes File, Edit, View, Run, Tools, Commands, and Help. The toolbar contains various icons for file operations and execution. The main window displays a text editor with a "Deck" tab. The code is color-coded and includes line numbers from 1 to 27 on the left margin. The code content is as follows:

```

1 |# (c) Silvaco Inc., 2015
2 |go athena
3 |
4 |#
5 |line x loc=0.0 spac=0.1
6 |line x loc=0.2 spac=0.006
7 |line x loc=0.4 spac=0.006
8 |line x loc=0.6 spac=0.01
9 |#
10|line y loc=0.0 spac=0.002
11|line y loc=0.2 spac=0.005
12|line y loc=0.5 spac=0.05
13|line y loc=0.8 spac=0.15
14|#
15|init orientation=100 c.phos=1e14 space.mul=2
16|#
17|#pwell formation including masking off of the nwell
18|#
19|diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
20|#
21|etch oxide thick=0.02
22|#
23|#P-well Implant
24|#
25|implant boron dose=8e12 energy=100 pears
26|#
27|#

```

Figure 3-14 Margin and Line Numbers Enabled



The screenshot shows the DeckBuild application window with the title bar "/export/data1/t". The menu bar includes File, Edit, View, Run, Tools, Commands, and Help. The toolbar contains various icons for file operations and execution. The main window displays a text editor with a "Deck" tab. The code is color-coded but lacks line numbers and a left margin. The code content is as follows:

```

# (c) Silvaco Inc., 2015
go athena

#
line x loc=0.0 spac=0.1
line x loc=0.2 spac=0.006
line x loc=0.4 spac=0.006
line x loc=0.6 spac=0.01
#
line y loc=0.0 spac=0.002
line y loc=0.2 spac=0.005
line y loc=0.5 spac=0.05
line y loc=0.8 spac=0.15
#
init orientation=100 c.phos=1e14 space.mul=2
#
#pwell formation including masking off of the nwe
#
diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
#
etch oxide thick=0.02
#
#P-well Implant
#
implant boron dose=8e12 energy=100 pears

```

Figure 3-15 Margin and Line Numbers Disabled

Figure 3-16 shows the entry of the Toolbars menu with its default settings.

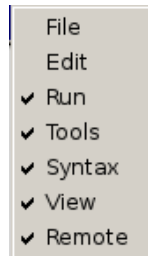


Figure 3-16 Default Toolbar Settings

In the following description of the toolbar functions, the default settings with only the **Run**, **Tools**, **Syntax** and **View** toolbars enabled was used.

3.5 Running Deck

In the following section, all the actions related to running a deck are described.

Figure 3-17 shows a screenshot of the Toolbar.



Figure 3-17 DeckBuild Toolbar

Here are the actions related to running a deck.

- **Save and Run/Continue** – Save the deck then start the simulation or continue from a Stop point
- **Run/Continue** – Starts a simulation or continues from a Stop point.
- **Pause** – Pauses a simulation. Use this if you want to temporarily relieve the CPU but don't want to terminate the simulation. Pausing will immediately “freeze” the simulation at the very line that is being executed. The simulator will be put into sleep mode by sending a UNIX STOP signal. When a simulation is paused, you can resume it by clicking **Pause** again. Unpausing will have the effect of sending a UNIX CONT signal.
- **Stop** – Stops the execution but does not kill the simulator. The **Stop** function will first wait for the simulator to finish the current line and then it will put the simulator into “Waiting”. The effect is the same as if you had defined a Stop point at the line where the simulator is stopped.
- **Kill** – When you invoke the kill action, then the simulator is killed immediately. It is not waited until the current line of deck has finished. No summary is displayed when the simulator terminates. This is a way of forcibly terminating the simulation.
- **Kill and Restart** – Same as **Kill** but will immediately restart from the top.
- **Quit** – Quits the simulation. This will first wait until the simulator has finished executing a line before sending a `quit` command to it. The simulator will display a summary of the resources that were used (i.e., CPU time) and will then exit.
- **Save and Next** – Save the deck then execute exactly one line of deck.
- **Next** – Executes exactly one line of deck.
- **Save and Run to Line** – Save the deck then execute up to the current location.
- **Run to Line** – Executes deck up to the cursor location.
- **Go to line** – Change the current line without executing deck.
- **Set a stop at cursor location** – Defines a Stop point.
- **Enable/Disable history**
- **Open History actions** - Open the history actions dialog
- **Init from history at the cursor location** – Loads a history file into the simulator.
- **Start TonyPlot** – Saves a structure from the current line and load it into TonyPlot.
- **Start TonyPlot 3D** – Saves a 3D structure from the current line and load it into TonyPlot 3D.
- **Start MaskViews** – Starts the MaskViews program.
- **Start Sedit**
- **Start Devedit**
- **Start Devedit3D**

- **Disable/Enable inline TonyPlot and TonyPlot 3D calls** – Allows to ignore tonyplot statements, which appear in the deck.
- **Run the Athena to VictoryProcess deck converter (A2VP)**
- **Strip commented Athena commands** inserted from a previous run of A2VP
- **Open the main syntax dialog**
- Open the **variables tracking** pane
- Open the **Output files tracking** pane

The looks of the Toolbar change while running a simulation since not every action is available at all times. [Figure 3-17](#) shows the Toolbar when deck is actually being executed. Compare to [Figure 3-18](#). Some actions are disabled (e.g., **Run**), whereas others are enabled (e.g., **Kill**).



Figure 3-18 Run Functions Disabled

The very same functions that you find on the toolbar are also available in the **Run** and **Tools** menus ([Figure 3-19](#) and [Figure 3-42](#)).

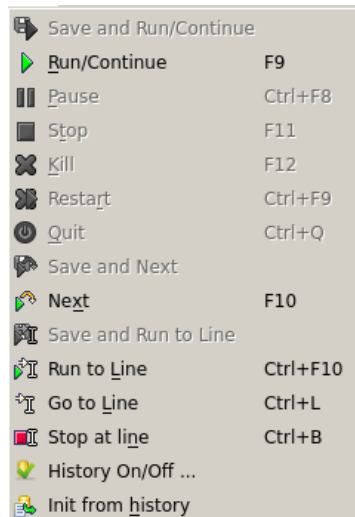



Figure 3-19 Run Menu

The following describes all the ways how to run a deck and how to halt the execution at any position in it. A very simple way to run the whole deck is by clicking the **Run** button (). This will start at the very first line and will go through the deck until either an error occurs or when the last line has been reached. You can also only execute subsets of the available deck or repeat running through certain portions of it.

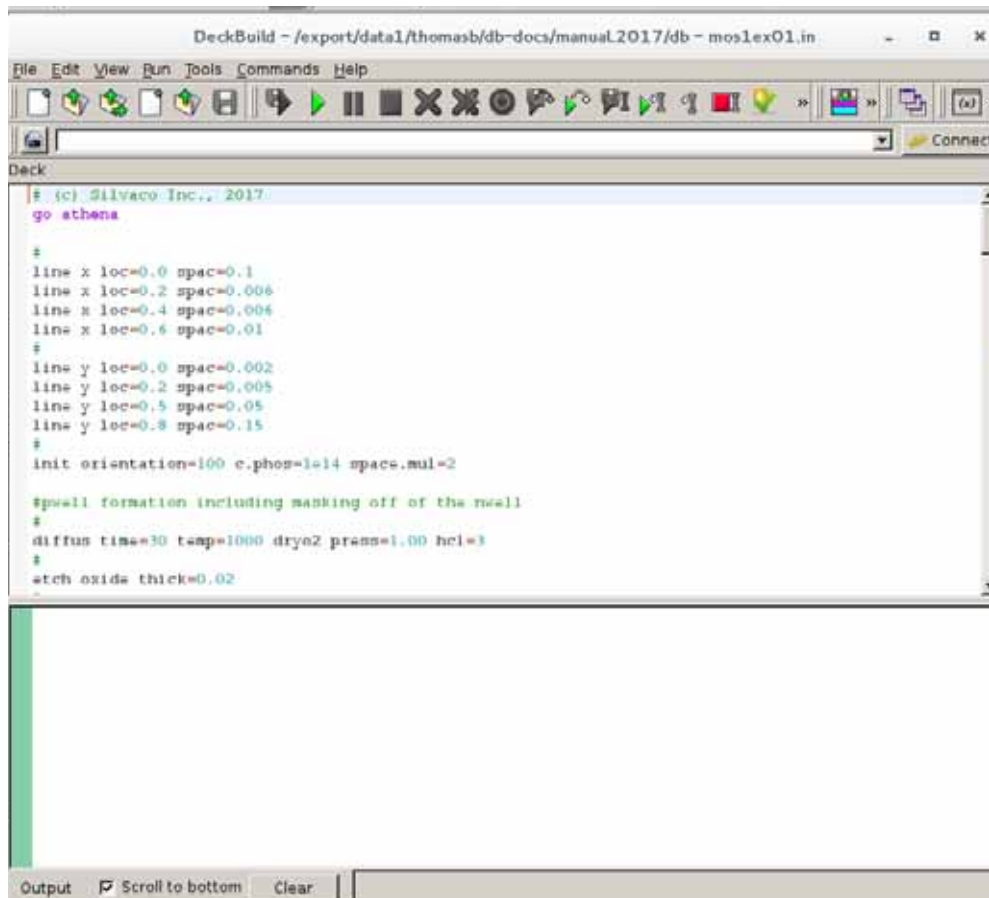


Figure 3-20 Loaded Deck Waiting to be Started

Figure 3-20 shows a screen-shot of DeckBuild with a deck loaded. The deck has not yet been executed. At the very top, the current line (with the copyright message as a comment) is indicated by a light blueish background. As we execute lines of deck, the current line will change.


You can choose to execute the deck line-by-line. This is called single stepping. To do this, click  (Next).

Figure 3-21 shows the result of one single step action. The current line has moved from line number 1 to line number 2 and the runtime output now contains a copy of the executed line. In this example, this is a comment, so no simulator was involved.

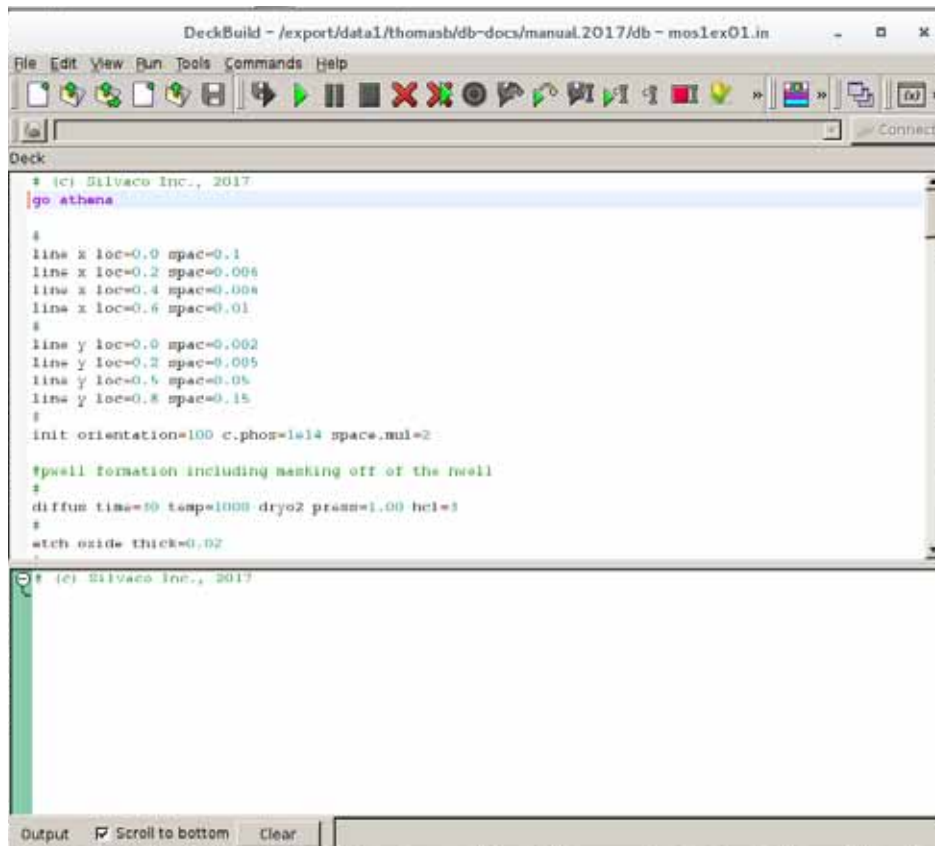



Figure 3-21 Executed Deck Line

Clicking  a second time will execute the line called `go athena`. This statement instructs DeckBuild to start the Athena simulator. Figure 3-22 shows the situation after the `go` line was executed. Note that the current line has advanced to line number 3 (an empty line). Text was also added to the Runtime Output pane. You can see the text is formatted according to where it originates from. All the black rendered text is actually a copy of the line of deck being executed. It originates from DeckBuild itself. The light grey rendered text originates from the simulator. By using different font settings, it is easier to recognize what part of the simulator output belongs to which deck statement.

Note: You can change the font settings for the runtime output in the Preferences panel.

The very last line of the runtime output in Figure 3-22 shows the following line:

```
ATHENA>
```

This is the prompt and it's an indication from the simulator that it is ready to accept (more) commands. Prompts are the way how DeckBuild and simulators communicate. DeckBuild will wait for the reception of a prompt before it sends the next line of deck to the simulator.

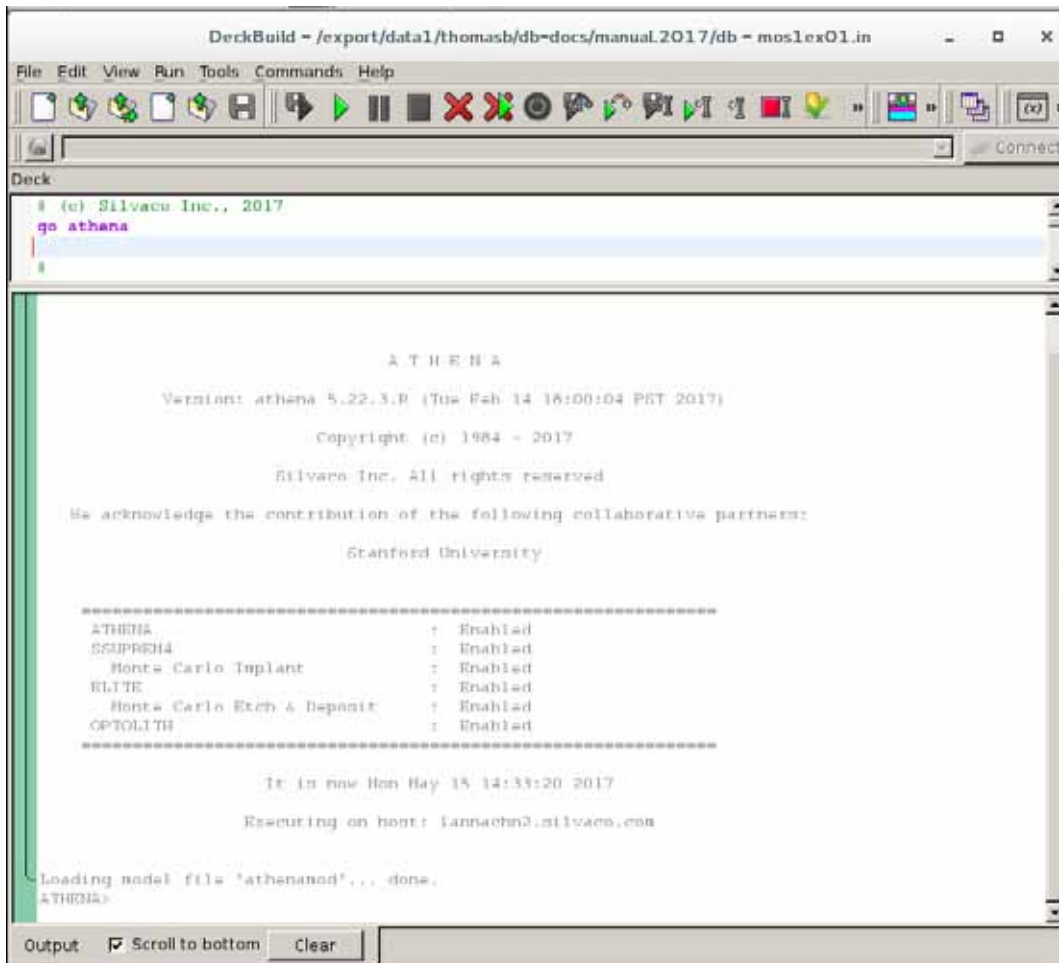
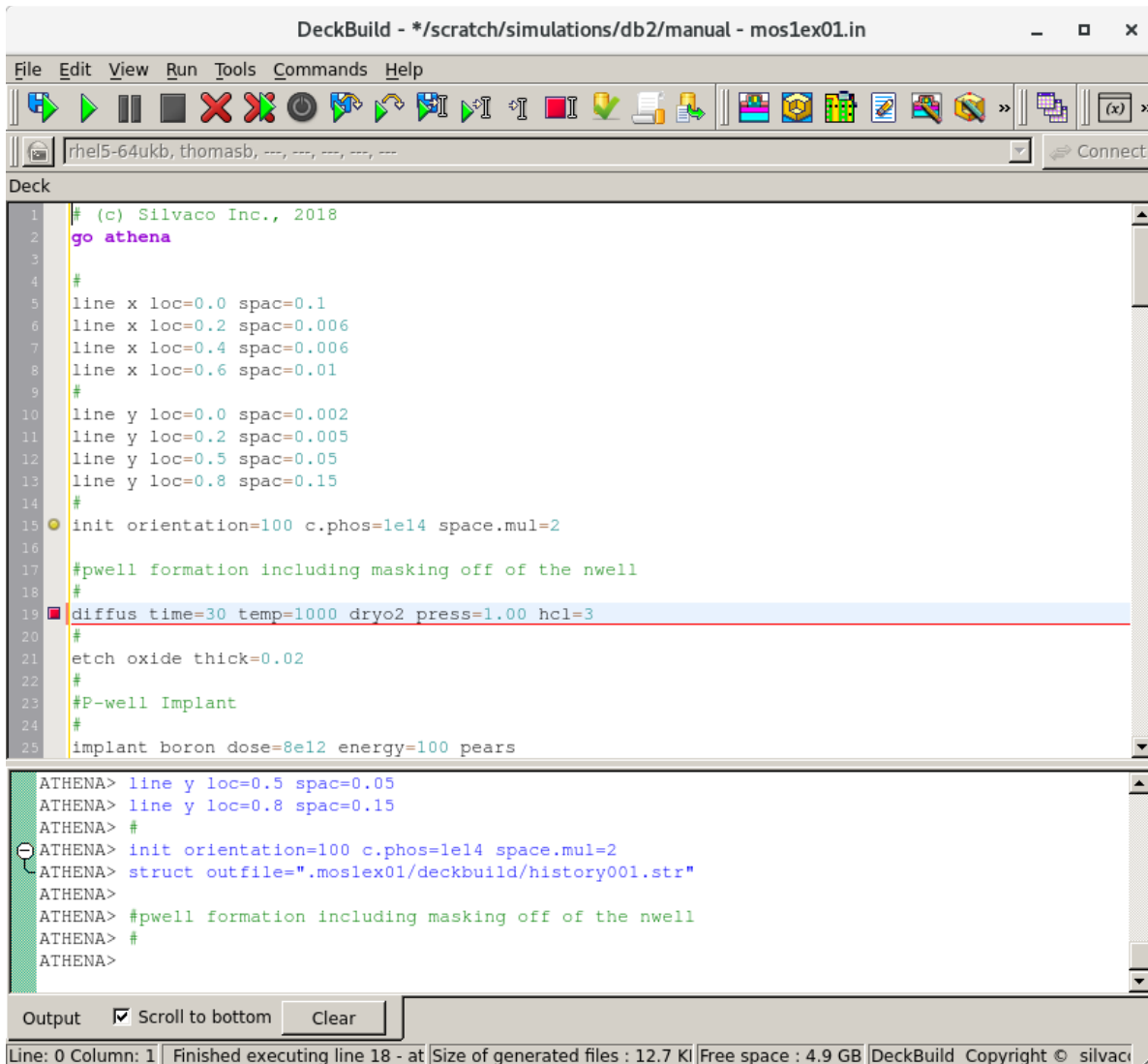


Figure 3-22 Started Athena Simulator

3.6 Stop Points

We will now demonstrate another way of executing the deck using “Stop points”. A Stop point marks a position in the deck where the execution will stop. You can define any number of Stop points. When executing the deck, only Stop points after the current line are considered. A Stop point that appears before the current line is ignored until you start over from a location before it.

Figure 3-23 shows a deck with two Stop points defined.



The screenshot shows the DeckBuild application window titled "DeckBuild - */scratch/simulations/db2/manual - mos1ex01.in". The interface includes a menu bar (File, Edit, View, Run, Tools, Commands, Help), a toolbar with various icons, and a status bar at the bottom. The main area is divided into two panes: "Deck" and "Output".

The "Deck" pane displays the following code:

```

1  # (c) Silvaco Inc., 2018
2  go athena
3
4  #
5  line x loc=0.0 spac=0.1
6  line x loc=0.2 spac=0.006
7  line x loc=0.4 spac=0.006
8  line x loc=0.6 spac=0.01
9  #
10 line y loc=0.0 spac=0.002
11 line y loc=0.2 spac=0.005
12 line y loc=0.5 spac=0.05
13 line y loc=0.8 spac=0.15
14 #
15 ● init orientation=100 c.phos=1e14 space.mul=2
16
17 #pwell formation including masking off of the nwell
18 #
19 ■ diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
20 #
21 etch oxide thick=0.02
22 #
23 #P-well Implant
24 #
25 implant boron dose=8e12 energy=100 pears

```

The "Output" pane shows the following text:


```

ATHENA> line y loc=0.5 spac=0.05
ATHENA> line y loc=0.8 spac=0.15
ATHENA> #
ATHENA> init orientation=100 c.phos=1e14 space.mul=2
ATHENA> struct outfile=".mos1ex01/deckbuild/history001.str"
ATHENA>
ATHENA> #pwell formation including masking off of the nwell
ATHENA> #
ATHENA>

```

The status bar at the bottom indicates: "Line: 0 Column: 1 | Finished executing line 18 - at | Size of generated files : 12.7 K | Free space : 4.9 GB | DeckBuild Copyright © silvaco".

Figure 3-23 Deck with Stop Points

If you click on , then the deck will be executed from the current line (line 3) up to the first Stop point in line number 19. Figure 3-24 shows the situation after reaching the Stop point. You can see that the current line has advanced to line number 19 and that the runtime output has changed showing the response of the simulator. The last line in the runtime output is again




a prompt. Click  again and the deck will be executed up to line number 28. Pressing  once more will execute the remainder of the deck since there are no more stop points defined.

Figure 3-24 also shows a little yellow bullet in the margin next to line 15. This is called the History feature (see Section 3.7 History Feature).

You can achieve a similar thing as with Stop points by clicking  (**Run to Line**).

To use this function, position the cursor in a line below the current line and click . DeckBuild will execute the entire deck from the current line down to the selected line and will stop there.

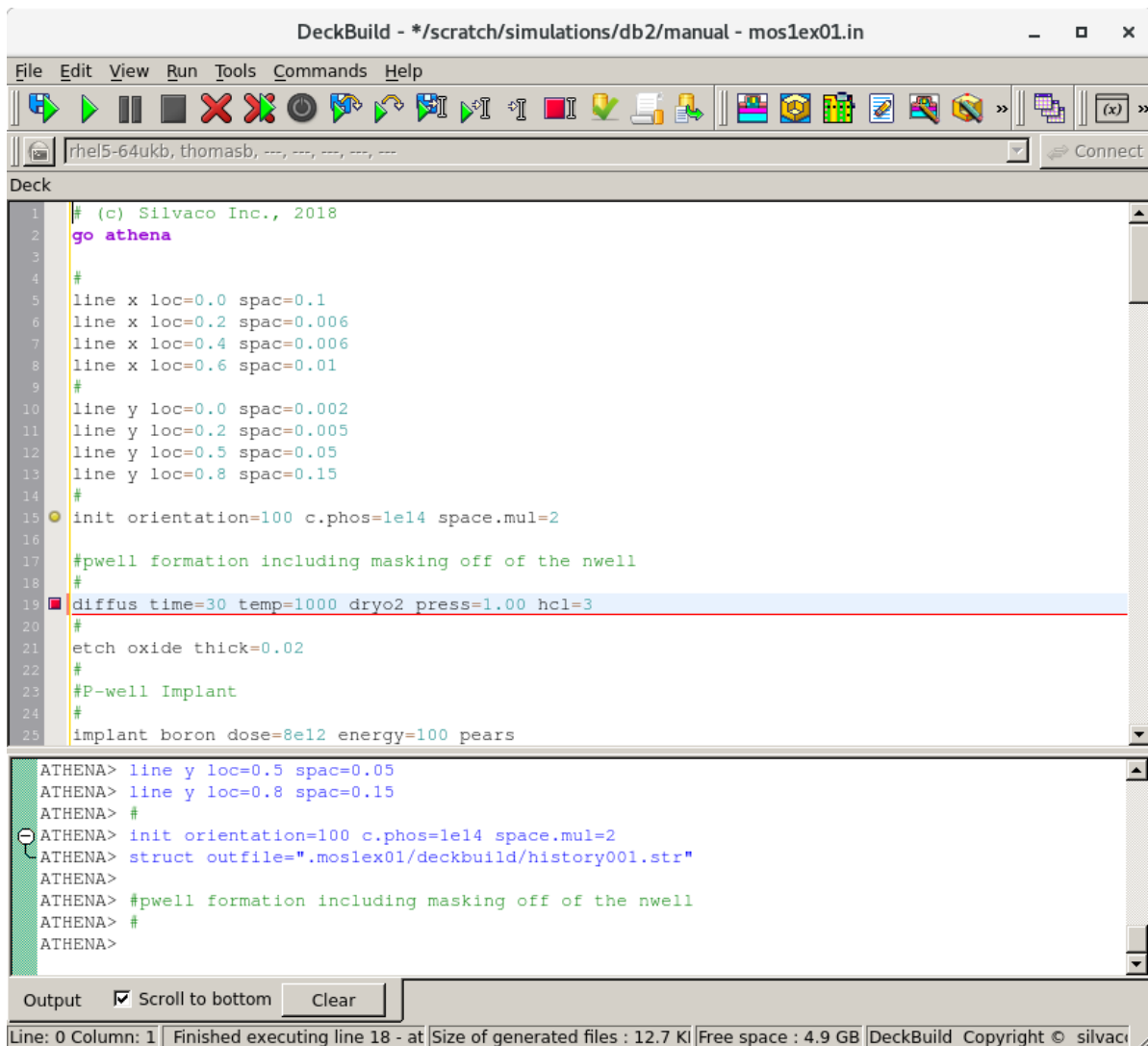


Figure 3-24 Halted by Reach of a Stop Point

3.7 History Feature

DeckBuild comes with a powerful feature called **History**. On the one hand side this allows you to reposition your current line to any previously executed point in the deck. On the other hand it is possible to use recorded history points to create PDF reports or movies to illustrate the simulation progress. The PDF report and movie generation feature will be further explained in [Section 3.8 PDF report and movie generation](#)

Deck lines with a history are indicated by a little yellow bullet on the left. [Figure 3-25](#) shows a deck, which has been run. You can see a total of five bullets on lines 54, 61, 66, 70, and 75. To see all the bullets, scroll up or down to see the other indicators.

Note: Not every line of deck has an associated history point. The Preferences section allows you to define how many history points are kept in total and how many lines without history will appear between two consecutive history indicators.

History points are implemented by saving the state of the simulator. This will usually mean a structure file is saved.

Note: Depending on simulator and type of deck, these files can be large so it is advisable to adapt the History settings to your particular situation.

To use a **History**, simply click on the associated bullet. [Figure 3-26](#) shows the situation right after the history bullet in line number 66 (`implant...`) was clicked. You can see that the current line was set to line number 67. Additionally, the runtime output indicates that the simulator was initialized from the following file:

```
ATHENA> init infile=.mos1ex01/deckbuild/.history023.str
```

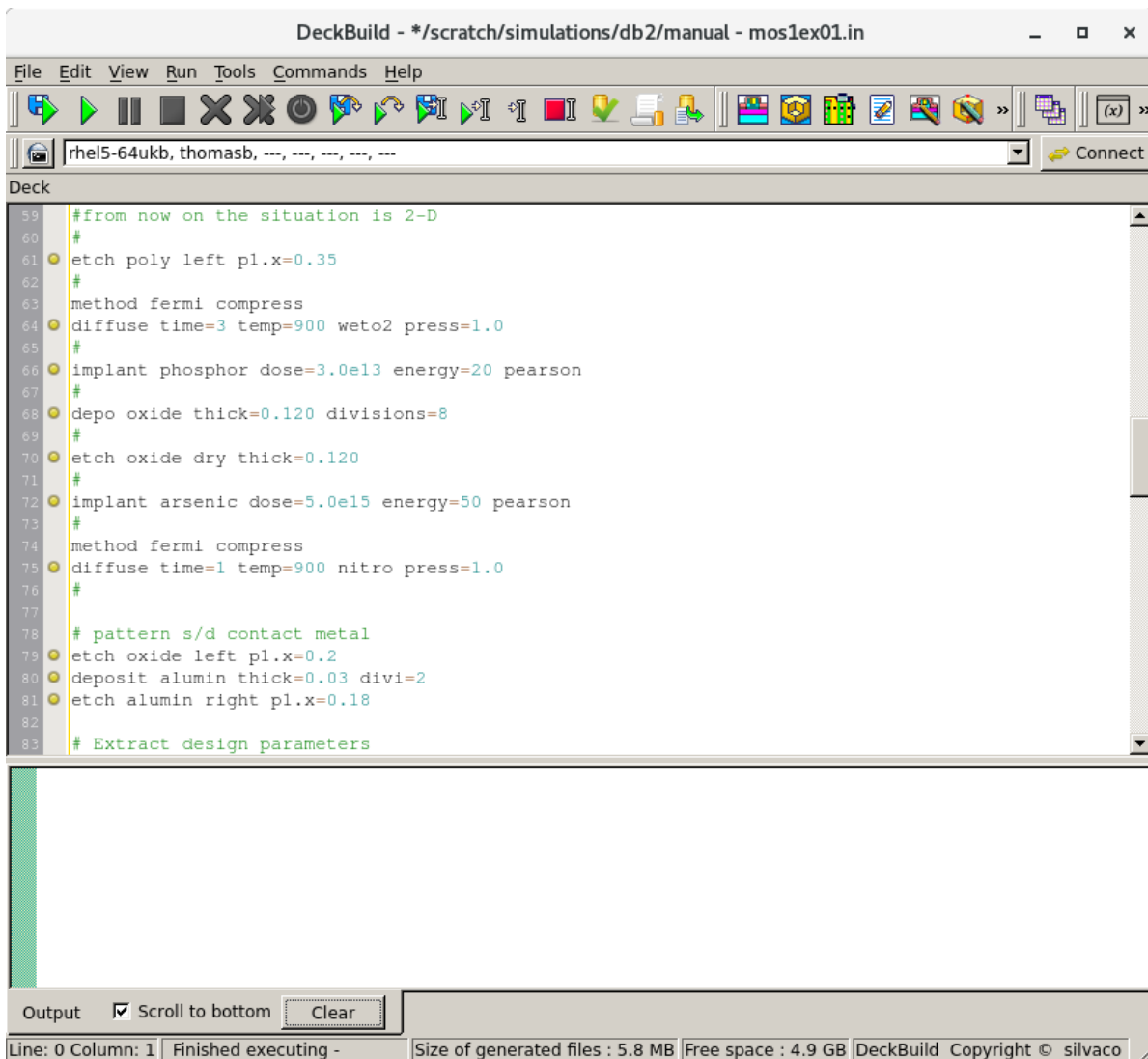
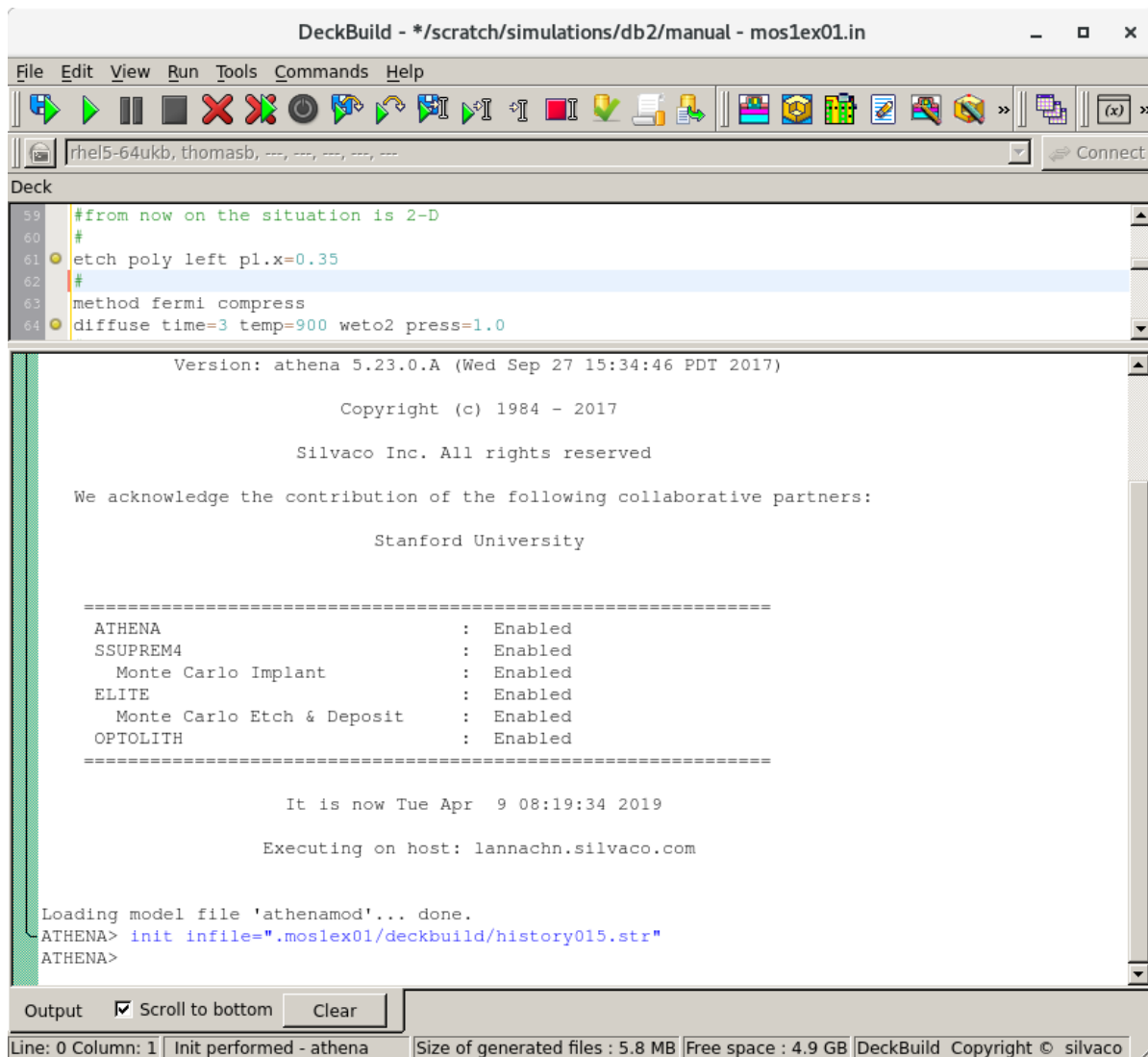


Figure 3-25 Deck with History Indicators

If the simulator had not already been running, it will be started by executing a corresponding `go` statement.

Note: The situation after loading a history file is the same as if you had reached this position in the deck by running it from top up to this point.

Note: Not every simulator supports **History**. Currently, The Athena and Victory Process simulators are capable of saving and loading history files.



DeckBuild - */scratch/simulations/db2/manual - mos1ex01.in

File Edit View Run Tools Commands Help

rhel5-64ukb, thomasb, ---, ---, ---, ---, ---

Deck

```

59 #from now on the situation is 2-D
60 #
61 etch poly left pl.x=0.35
62 #
63 method fermi compress
64 diffuse time=3 temp=900 weto2 press=1.0

```

Version: athena 5.23.0.A (Wed Sep 27 15:34:46 PDT 2017)

Copyright (c) 1984 - 2017

Silvaco Inc. All rights reserved

We acknowledge the contribution of the following collaborative partners:

Stanford University

=====

ATHENA	:	Enabled
SSUPREM4	:	Enabled
Monte Carlo Implant	:	Enabled
ELITE	:	Enabled
Monte Carlo Etch & Deposit	:	Enabled
OPTOLITH	:	Enabled

=====

It is now Tue Apr 9 08:19:34 2019

Executing on host: lannachn.silvaco.com

Loading model file 'athenamod'... done.

ATHENA> init infile=".moslex01/deckbuild/history015.str"

ATHENA>

Output Scroll to bottom

Line: 0 Column: 1 Init performed - athena Size of generated files : 5.8 MB Free space : 4.9 GB DeckBuild Copyright © silvaco

Figure 3-26 History Loaded

3.8 PDF report and movie generation

In this chapter we will emphasize two new features of the DeckBuild deck editing environment. These are the movie creation on the one hand and the creation of PDF reports on the other. This version of Deckbuild allows you to use recorded history points to prepare movies and PDF reports of a simulation .

The history feature does not only allow you to restart a previously run simulation at arbitrary points in the deck (Section 3.7 History Feature), it is also possible to hook up and run post-processing scripts to history points. The scripts allow for converting the simulation state into PNG images or to prepare cuts through the structure and save them as PNGs. Basically, any other Silvaco or system tool can be utilized this way.

Note, that you need to use the XML file format to make the history status persistent, otherwise all history information is lost when deckbuild terminates.

3.8.1 History Scripts


In order to use the saved simulation status for either the movie or the PDF report generation a post-processing step must be performed on the saved data. This is done by attaching and running a script to the history data. Note, that the default action is to not assign or run any scripts. While this will still allow you to load a history point as described above no further actions will be possible. To assign and run scripts, you need to use the history actions dialog. It is opened by clicking on the second icon from the right (), on the toolbar of the main DeckBuild window.

Figure 3-27 shows the available controls for assigning and running history scripts. The dialog presents a list of all history items that have been created by the simulation so far, and gives additional information for every history item. In the very left column, the line number of where in the deck the history was recorded is indicated. It corresponds to the line numbers as shown in the main window (see Figure 3-25) The second column from the left shows the status of a history item.

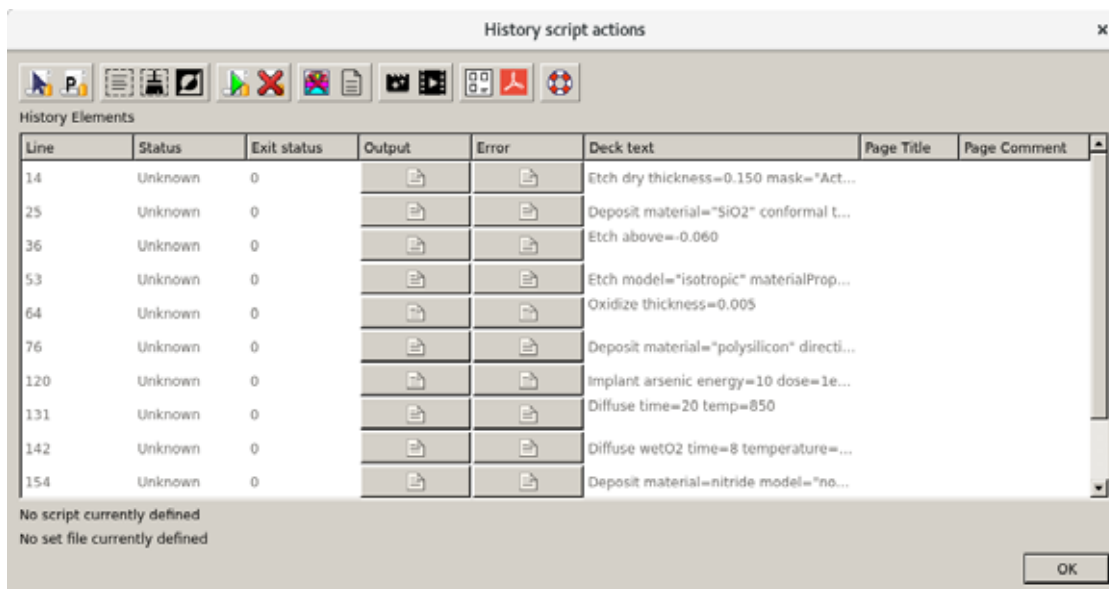



Figure 3-27 History dialog showing available history points

When the dialog is first opened and no script had been assigned before, the status for all listed history items is 'Unknown'. This will change as soon the script executes. The third column from the left gives the exit status of the script once it has terminated. A value not equal to 0 will indicate that a problem during script execution has occurred. The next two columns offer buttons, which allow you to look into the console output of the script (stdout and stderr channels). This is useful to get an idea of why a script may have failed or what information it produced on the console. The next column - 'Deck text' - shows the line of deck for which the history was recorded. You can use it to roughly identify the part of your simulation flow a history item belongs to. The final two columns: 'Page Title' and 'Page Comment' will be used by the PDF report generator. They will be explained further down in this section.

Note, that the content of the dialog is updated as the list of available history items grows.

The first action to take is to assign a script for execution. To do so you need to bring up the history scripts dialog. It is opened by clicking the first icon from the left () in the history actions dialog toolbar. [Figure 3-28](#) shows the scripts dialog populated with all available scripts. The scripts come pre-installed with the deckbuild tool. Every script has a short description and you will be able to edit parameter values should the script require parameters to fill in. If you are satisfied with your choice please close the dialog by clicking Ok. You will notice that the status 'No script currently defined' as shown in the status line of [Figure 3-27](#) will change to the name of the selected script. Everytime you re-open the history script dialog to chose a different script, the status will immediately indicate the newly chosen script as soon you Ok the dialog.

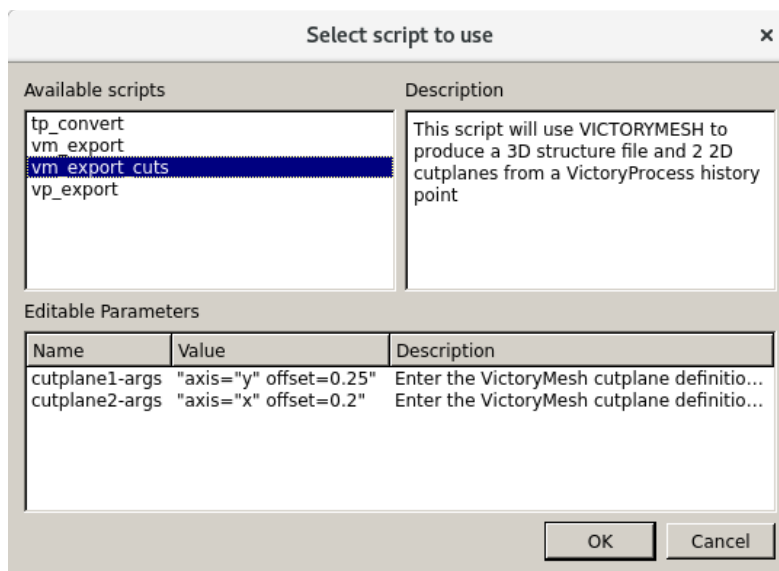



Figure 3-28 Choose script to run

Once a script was assigned the items in the history actions dialog become selectable. To run a script for a particular history item, the item must be selected (highlighted.) After selecting the items to run the 'Queue selected' button () must be clicked. [Figure 3-29](#) shows that 5 items have been selected and 4 scripts are actually executing (Status set to 'Running'). The 5th item is waiting for any of the other jobs to finish. The execution of jobs uses a built-in queuing system, which allows for a certain number of jobs (4 in this example) to run in parallel. Jobs

are executed independently from the simulation of the deck. In other words, the simulation will not be interrupted or influenced by the history script in any way.

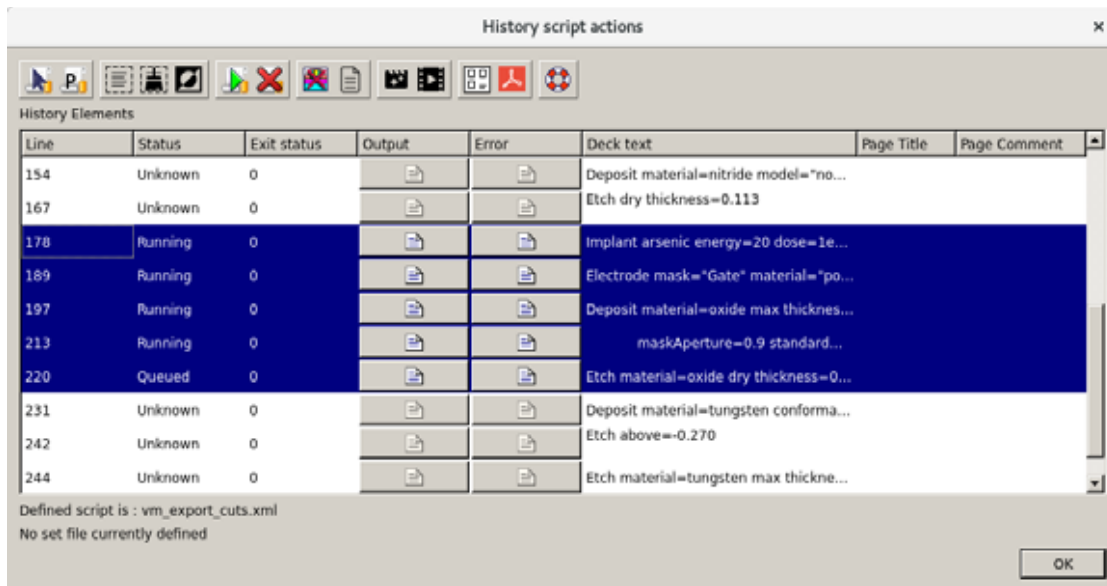


Figure 3-29 Executing history scripts

Note, that there are two ways of how a script gets queued for execution.


The first way is to use the selection mechanism and clicking on the 'Queue selected' button as described above. This method must be used when scripts are to be run on a simulation, which was already finished. The second method will automatically queue scripts while the simulation is actually executing. A job will be queued for every new history item to appear. This requires that a script has been defined prior to a history point appearing. If you assign a script while the deck is executing then all history items existing prior to the script assignment will not have jobs queued, all items that appear after the script was assigned will have jobs queued to run the assigned script. You can also combine the two methods and select/assign history items that would be missed out otherwise.

If you want to terminate a 'Queued' or 'Running' script you have to select the items to terminate and click on the 'Clear selected' button (✖).

3.8.2 Movie creation

In order to create a movie you must assign a script, which creates at least one PNG image of the structure. The task of creating a PNG image typically involves the use of the VictoryMesh and TonyPlot tools. The history script `vm_export_cuts.xml` will convert the VictoryProcess history point into three different structure files, one birds view and two cutplanes according to the cutplane definition given when the script was assigned. You can change the parameters of a script by clicking on the 'Edit script parameters' button (P). This will open the parameter dialog shown in [Figure 3-30](#).

The `vm_export_cuts.xml` script defines two cutplanes to create cuts through the structure. The cuts will be shown in the PDF report along with the 3D view (birds view). [Figure 3-30](#) shows two parameters named `cutplane1-args` and `cutplane2-args` of the script `vm_export_cuts.xml`. The column named 'Value' must contain the actual cutplane definition and must follow the syntax of the VictoryMesh cutplane command. In this example

two cutplanes along the ‘x’ and ‘y’ axis are defined. Note, that VictoryMesh allows you to define arbitrary cutplane locations. You can bring up the VictoryMesh help on the cutplane command by clicking the help icon () of the history dialog.

Note, that the `vm_export_cuts.xml` script defines cutplane parameters used by VictoryMesh during the creation of the structures. It is therefore necessary to rerun the script everytime the parameters are changed.

After the structure files were created TonyPlot will be used to read each one of them and create a corresponding PNG image.

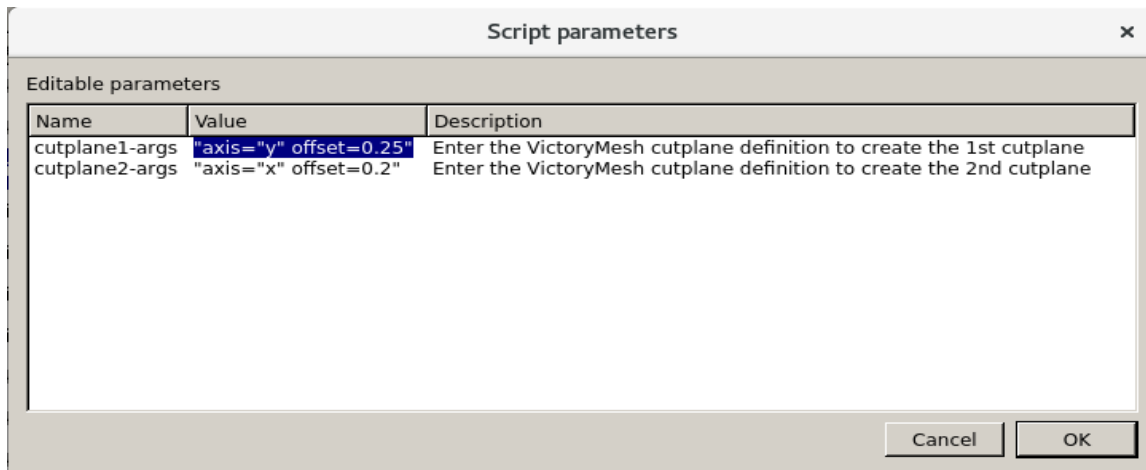



Figure 3-30 Edit script parameters

Once a script finishes without an error, this item can be included in the movie creation. You have the choice of creating up to three different movies out of the birds view and the two cutplanes. Use the ‘Setup movie’ button () to open the dialog as shown in [Figure 3-31](#) and choose, which movies you would like to create.

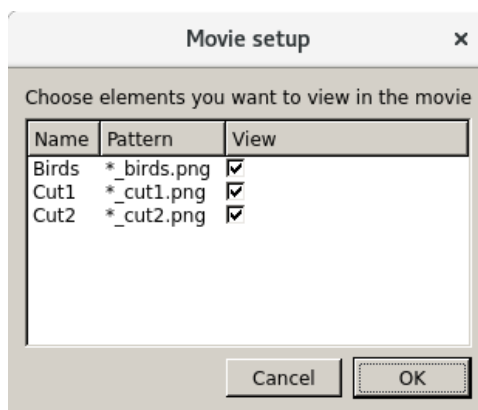




Figure 3-31 Selecting movies to create

To generate the movies you have to click the ‘Generate movie’ button (). This will create up to three animated GIF files depending on your choice and will load them into a viewer. [Figure 3-32](#) shows Deckbuild’s builtin movie viewer window.

Note, that a successful creation of an animated GIF requires that a PNG image was created by the history script.

Should you have chosen a different script, or in case you had forgotten to queue/run the scripts, or if there was an error etc., then no frame is created for the given history item. It is advisable to check the runtime output to further investigate the cause of the problem.

In case you want to add/remove frames from the movie you have to close the movie dialog and change the selection (the highlighted history items). Everytime you hit the 'Generate movie' button () the movie files are re-created (overwritten). The movie file is found in a hidden sub-directory of the experiment directory. In this example the directory is `\.vpx02/deckbuild` and the movie files are called `Animated_0.gif`, `Animated_1.gif` and `Animated_2.gif` respectively. You may want to copy the files to another location in you want to keep them permanently. When deckbuild exits it asks you to cleanup any generated data. This would include the movie (and also the PDF report) should you opt to clean the data.

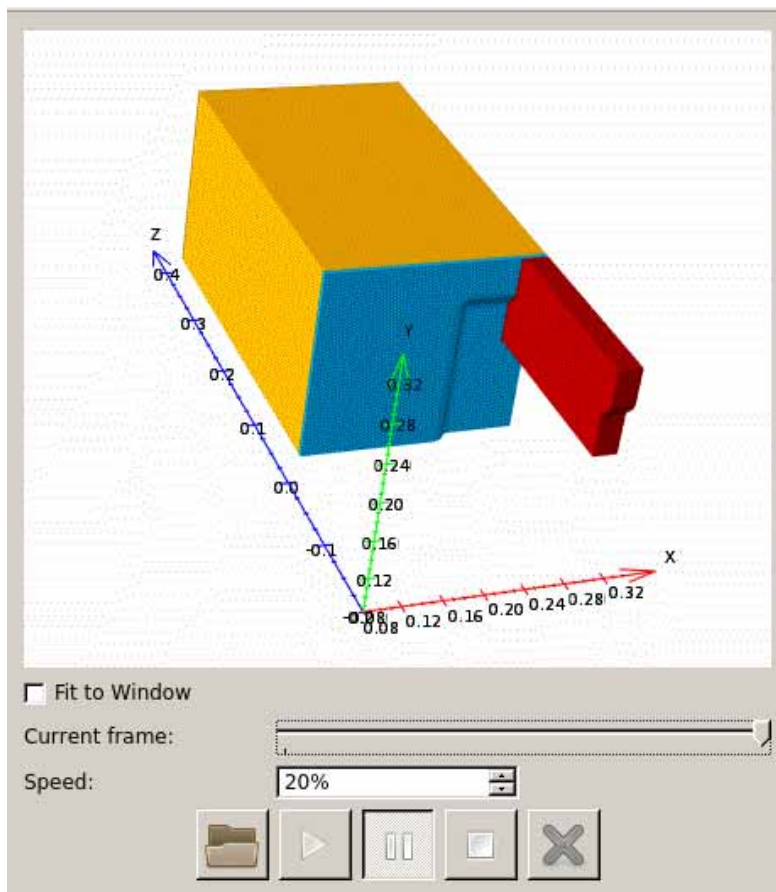



Figure 3-32 Movie viewer

3.8.3 Browsing through history points

Instead of creating a movie out of all available PNG images, you do have the option of browsing structure files of the history items using the TonyPlot tool. Click on the TonyPlot icon () in the history actions dialog to open TonyPlot and load the structure file of the active history item. Once TonyPlot was started this way, you can then navigate to a different history item and the corresponding structure file (should it be existing) will be loaded into the running TonyPlot program. You will also see that the cursor position in the main deck window will jump to the deck statement, which had resulted in creating the history action.

Click on the TonyPlot icon again or close TonyPlot to stop browsing the history items this way.

Please note that the file to be loaded will be the birds view.

Should you only be interested in browsing through all available history points you can choose a different script, to only create a single structure file. The `vm_export.xml` script will only export the VictoryProcess data into a structure file but will not compute any cutplanes. This will significantly speed up the export process.

3.8.4 PDF report creation

Additionally to creating up to three individual movies, you do have the choice of creating a PDF with one page per history point. Each page contains four areas containing up to three images (birds view and two cutplanes) and a text box. Deckbuild allows you to arrange the locations of the four content boxes to your needs.

DeckBuild offers controls to assign the content boxes on a template page to define the layout of the page. The boxes can be occupied with images or text. Images need to follow a certain naming scheme and need to have been created by the script. Content for the text box is either taken from the deck or can be entered manually on a per page level.

The next step is to define the page layout. To do so, you need to click on the ‘Page layout for PDF’ button (☐☐). This will bring up the layout template dialog. [Figure 3-33](#) shows the page template dialog with two positions, ‘view’ and ‘cut 1’ already defined. You can drag the remaining two positions, ‘text’ and ‘cut 2’ into the page preview or re-arrange items in the preview by dragging them onto each other. When done, please ‘Ok’ the dialog.

As a final step please add information in the ‘Page title’ and ‘Page comment’ columns (see [Figure 3-29](#)) and hit the ‘Generate PDF’ button (📄). This will start the generation process and open the produced PDF file.

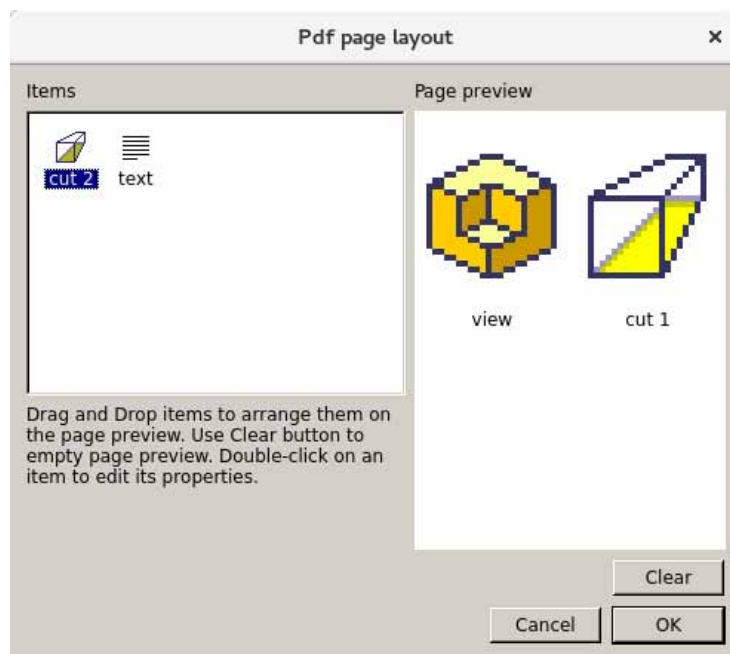


Figure 3-33 Setup PDF layout

3.9 Go to Line

Apart from using the [History Feature](#) described in the previous section, you can also change the current line of execution without restoring the original state. For instance, you can use this to rerun a particular line of deck without the need to copy/paste it in the editor.

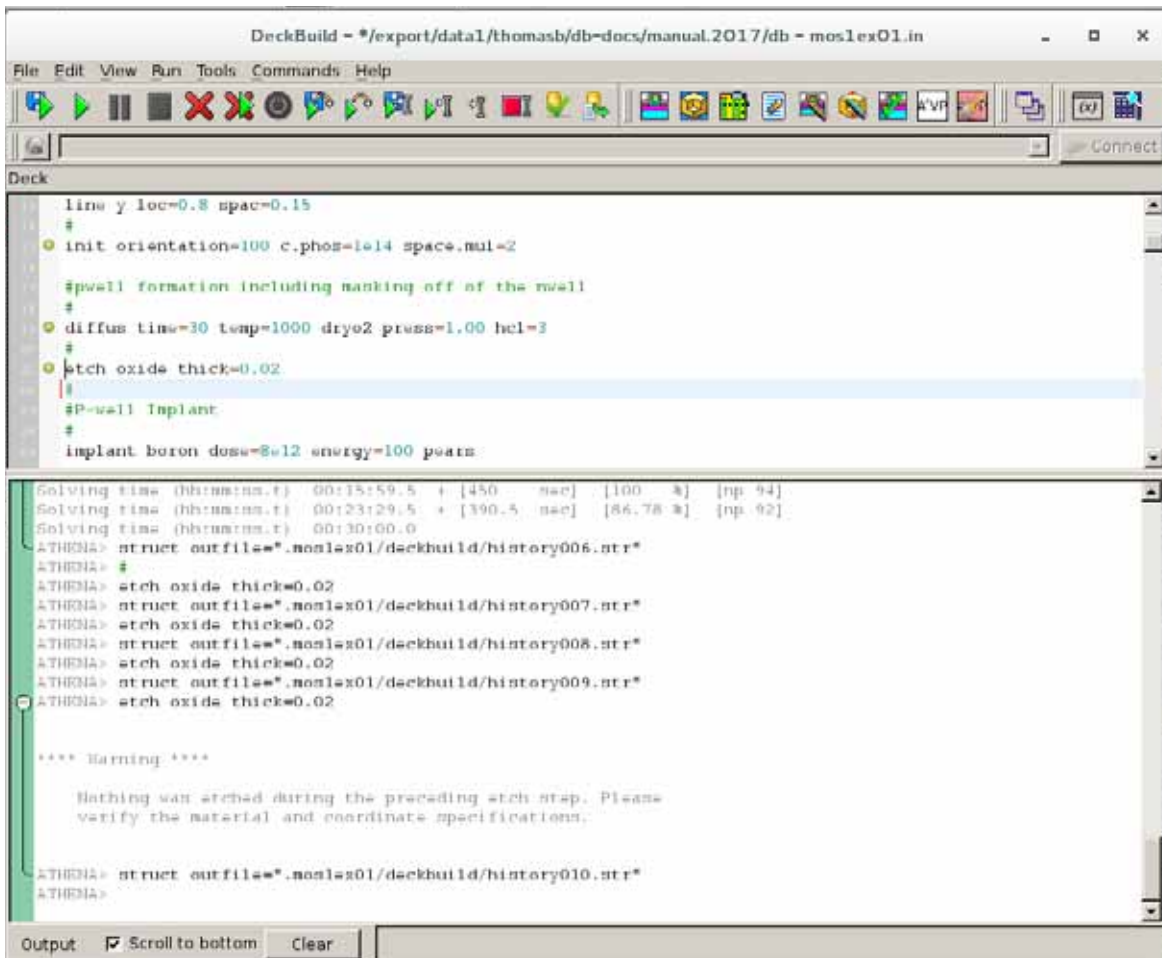



Figure 3-34 Go to Line Feature

In [Figure 3-34](#), the runtime output shows that deck line number 21 (etch) was executed four times. The fourth execution led to a simulator error.

3.10 Tracking Variables

Deckbuild provides a powerful means of keeping track of any variables, extracts and output files that are created during a simulation. Everytime a set statement or an extract is executed the corresponding variable is displayed in the variables pane.

You can toggle the variables pane by clicking on the variables icon  of the main toolbar. [Figure 3-35](#) displays the deckbuild window with the variables pane enabled. You can see that the pane is populated with variable names and values. In braces, the line number of the corresponding set or extract statement is given. The first entry in the variables pane is gateox, which corresponds to the extract statement in line 50 of the deck:

```
extract name="gateox" thickness oxide mat.occno=1 x.val=0.05
```

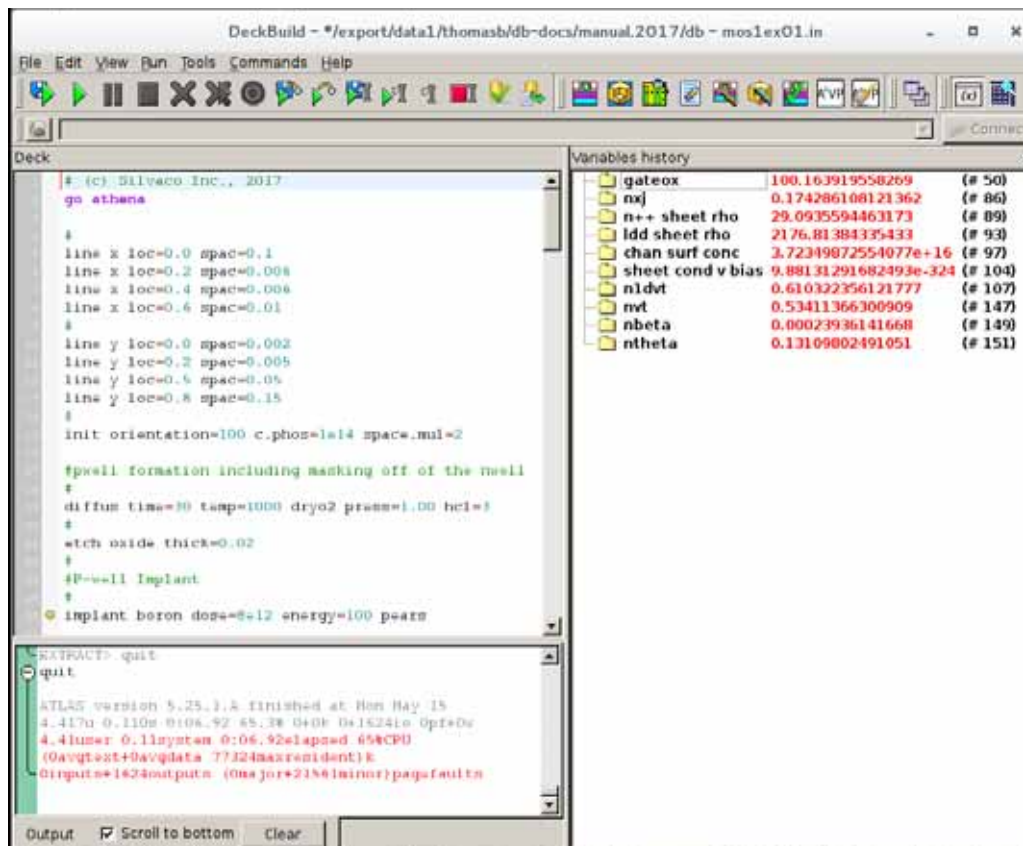


Figure 3-35 Variables Pane Enabled

Clicking on the value of a variable allows you to change its value. This is useful in debugging decks with lots of set statements. The change is effective immediately and will influence any statement, which takes as input the changed variable. [Figure 3-36](#) displays the situation after a deck containing three set statements is executed. The variables are assigned values according to the statements.



Figure 3-36 Executing set statements

If you single step through this deck you can now change a variable value before continuing to run the deck. Figure 3-37 illustrates the situation where execution has stopped after the first set statement and the value of x was changed from 10 to 200. Figure 3-38 shows the situation after the remaining deck was executed. You can see that variables y and z now have different values as shown in Figure 3-17.

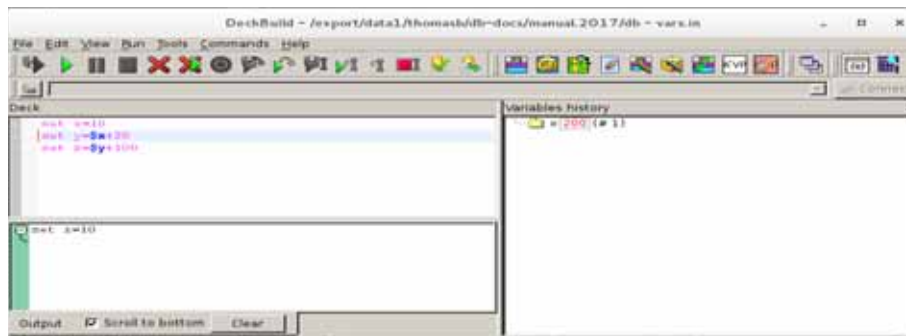


Figure 3-37 Edited Variable x



Figure 3-38 Influencing values of variables

3.11 Tracking Output Files

Deckbuild offers a view at all files available in the directory where the deck is executed. This will include all created structure files except files created by the history feature. History files are considered as deckbuild internal files (i.e. not created by an explicit deck statement)

You can toggle the outputs pane on and off by clicking on the outputs icon.



Figure 3-39 displays the deckbuild window with both the variables tracking and the outputs pane enables. Note, that both panes can be enabled separately allowing you to optimize the available window space.

At the very top of the outputs pane there is a filter available, which can be used to limit the amount of shown files.

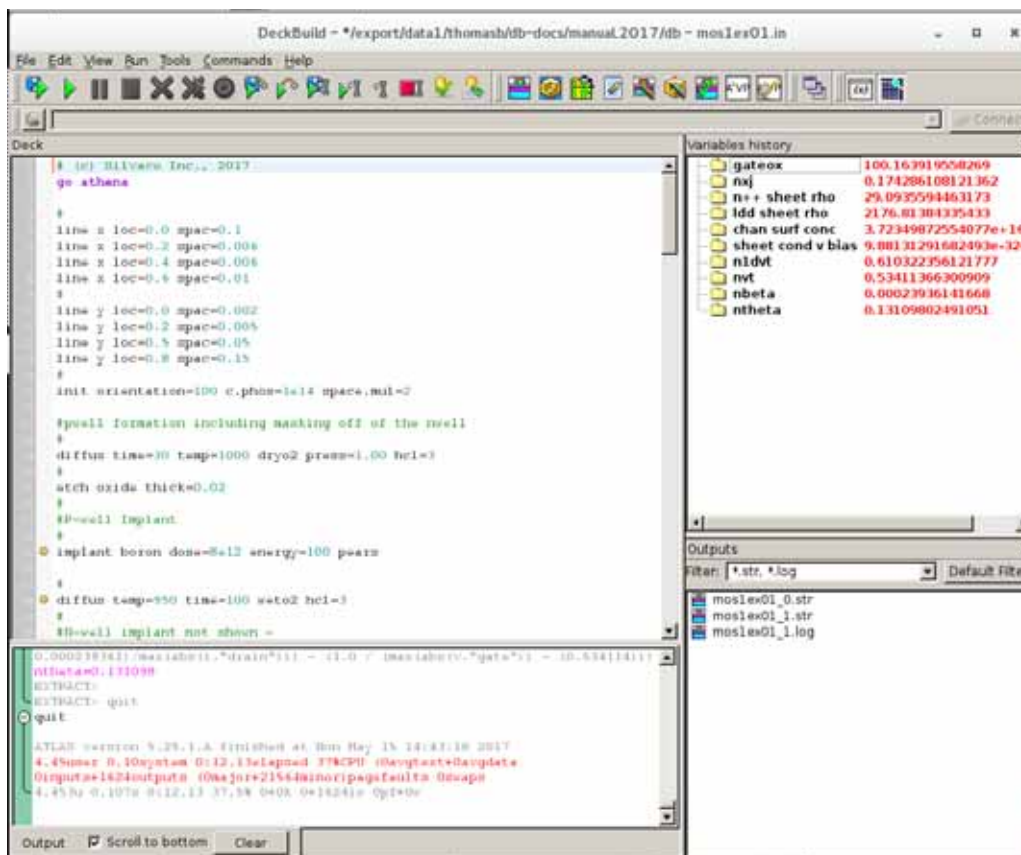


Figure 3-39 Outputs and variables panes enabled

The outputs pane is not limited to only viewing a list of all created files, you can also use it to directly load a file into TonyPlot or any other viewing application. Figure 3-40 displays the contextual menu, which opens upon right-clicking a displayed file.

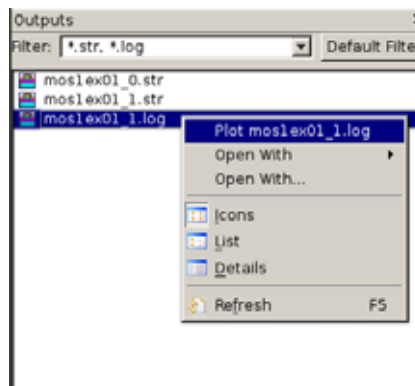



Figure 3-40 Plotting from the Outputs pane

3.12 Tracking Resource Usage

Deckbuild allows you to monitor the resource usage of the running simulator. The tracked information consists of the amount of used RAM, the disk I/O read and disk I/O write operations.

You can toggle the resource usage pane on and off by clicking the resource icon 

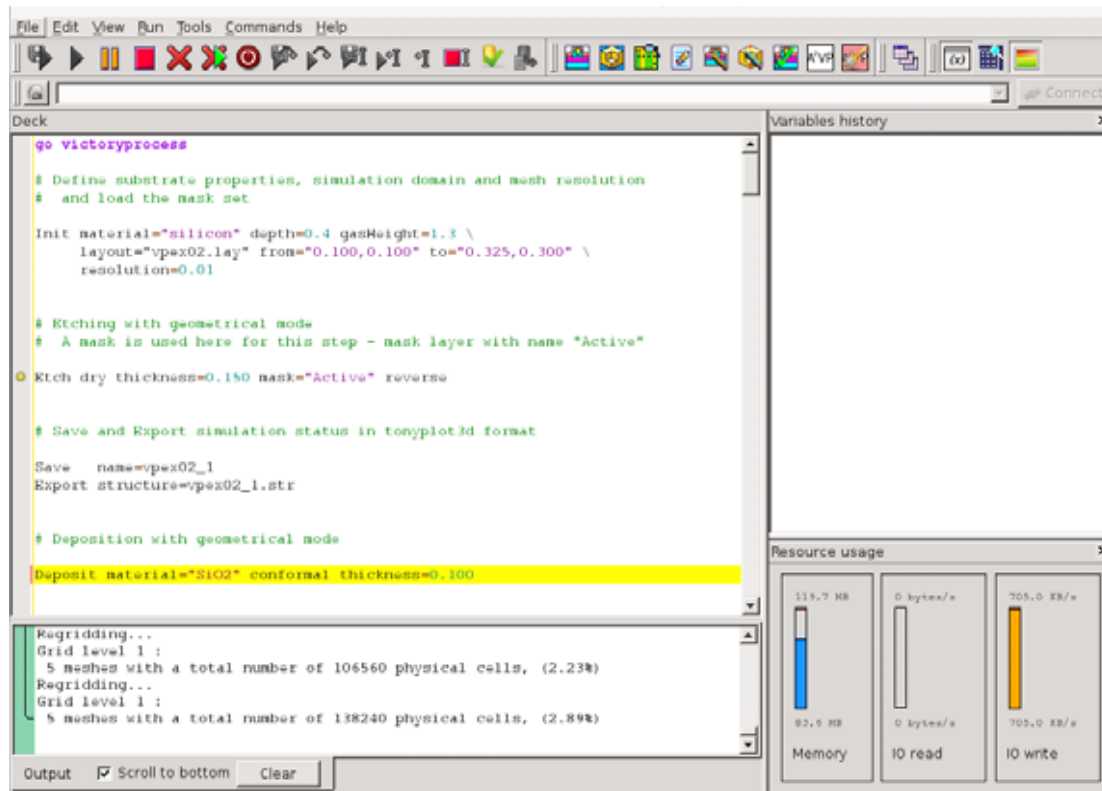


Figure 3-41 Resource usage pane turned on

Figure 3-41 shows a running simulation with the resource usage tracker turned on. You can see that the memory consumption at the time the screen-shot was taken was around 85MB. The number at the top of the bar indicates the maximum amount of RAM that has ever been used throughout this simulation. Next to the bar displaying the RAM two more bars are shown. They indicate I/O read and I/O write respectively. A write rate of ~700kB/s is shown in this examples.

3.13 Tools Menu

The **Tools** menu offers a very convenient way to open various tools (Figure 3-42). Clicking on any of the icons will open the corresponding tool.

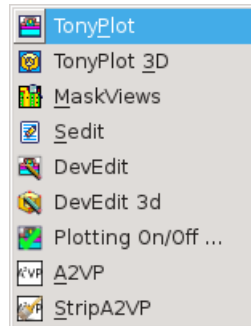



Figure 3-42 Tools menu

When the simulation is in the “Waiting” state, pressing  (**TonyPlot**) will save a structure file from the current line in the deck. It will then start TonyPlot and will load the saved file into it. If there is a History file, which had been saved on the line, then no extra file will be saved and the history file will be loaded into TonyPlot.

The same is true for TonyPlot 3D. A file is exported and loaded into TonyPlot 3D.

3.14 Edit Menu

Figure 3-43 shows the **Edit** menu. This menu contains the text editor functions. They are as follows:

- **Undo** – Undoes a previous editing operation.
- **Redo** – Redoes the effect of a previous Undo operation.
- **Copy** – Copies highlighted text into a buffer space.
- **Cut** – Removes the highlighted text and puts it into a buffer space.
- **Paste** – Pastes any text from the buffer space into the editor at the current position.
- **Clear** – Empties the editor window.
- **Select All** – Highlights all text in the editor in preparation for a **Copy/Cut** operation.
- **Find** – Searches for the desired text in the editor.
- **Find Next** – Continues to search for the desired text further down in the editor..
- **Find previous** – Continues to search for the desired text further up in the editor .
- **Replace** – Replaces found occurrences of text with the new desired text.
- **Preferences** – This opens the Preferences panel.

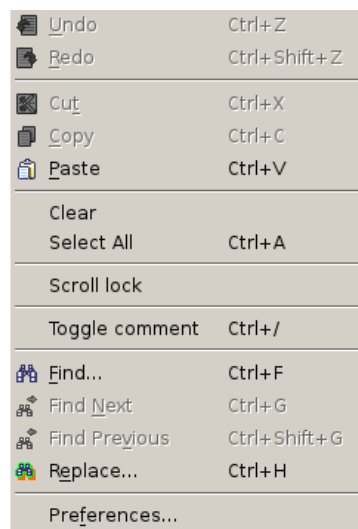


Figure 3-43 Edit Menu

3.15 Help Menu

Figure 3-44 shows the **Help** menu. This will open the DeckBuild User's Manual.



Figure 3-44 Help Menu

3.16 File Menu

Figure 3-45 shows the items from the **File** menu.

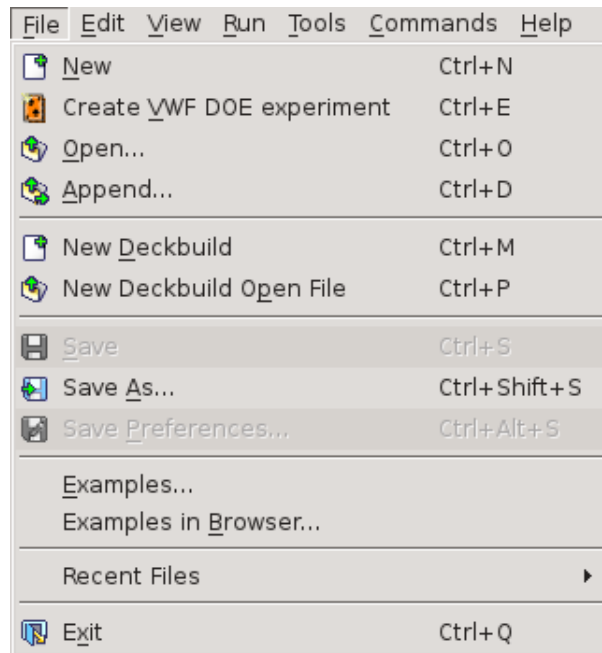


Figure 3-45 File Menu

- **New** – Clears the editor contents
- **Create VWF DOE Experiment** – Loads the deck together with all needed input files into VWF
- **Open** – Opens a file from disk
- **Append** – Opens a file and append at the end of the editor contents
- **New Deckbuild** – Opens a new empty deckbuild window
- **New Deckbuild Open File** – Opens a file in a new deckbuild window
- **Save** – Saves the editor contents
- **Save As** – Saves the editor contents to a given file
- **Save Preferences** - Preferences are normally saved when deckbuild exits; This point allows you to save the preferences right away, which may make sense when you start other deckbuild instances.
- **Examples** – Opens the examples dialog
- **Examples in Browser** - will open the system configured browser and display the example tree. Allows you to navigate through the available examples.
- **Recent Files** – Allows to open a file from a list of recently opened files
- **Exit** – Terminates deckbuild

3.17 Examples

There are more than 500 examples that are shipped with DeckBuild. To search for these examples, select **File**→**Examples**, and a dialog will appear (see [Figure 3-46](#)). The dialog is initially loaded with a hierarchical view at the examples. Select an example from the tree, or enter a search string.

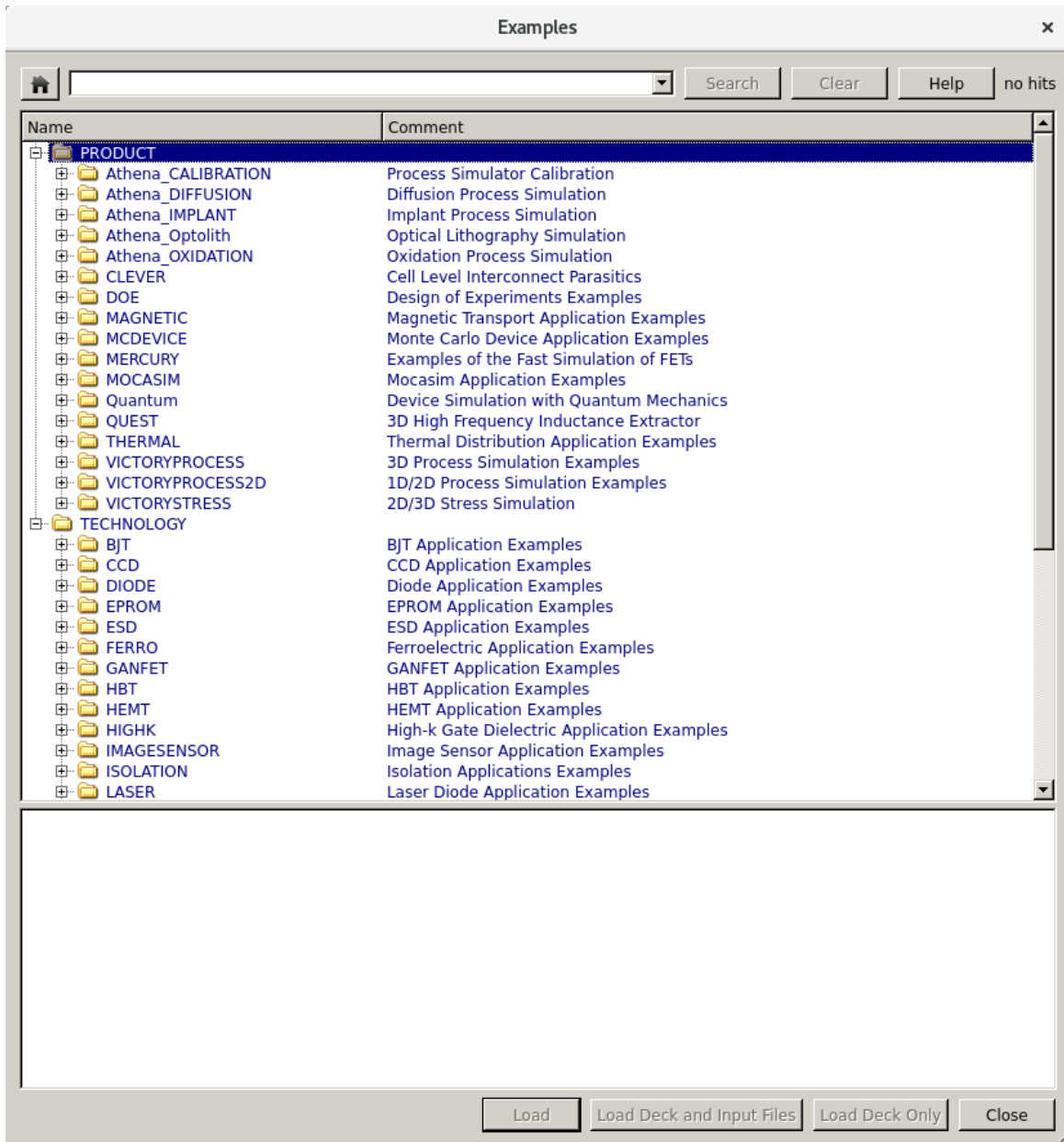


Figure 3-46 Examples Search Dialog

[Figure 3-47](#) displays the examples dialog with an example selected from the hierarchy. The selected example is **mos1ex01** from the **section MOS1**, which has been used in many examples throughout this manual.

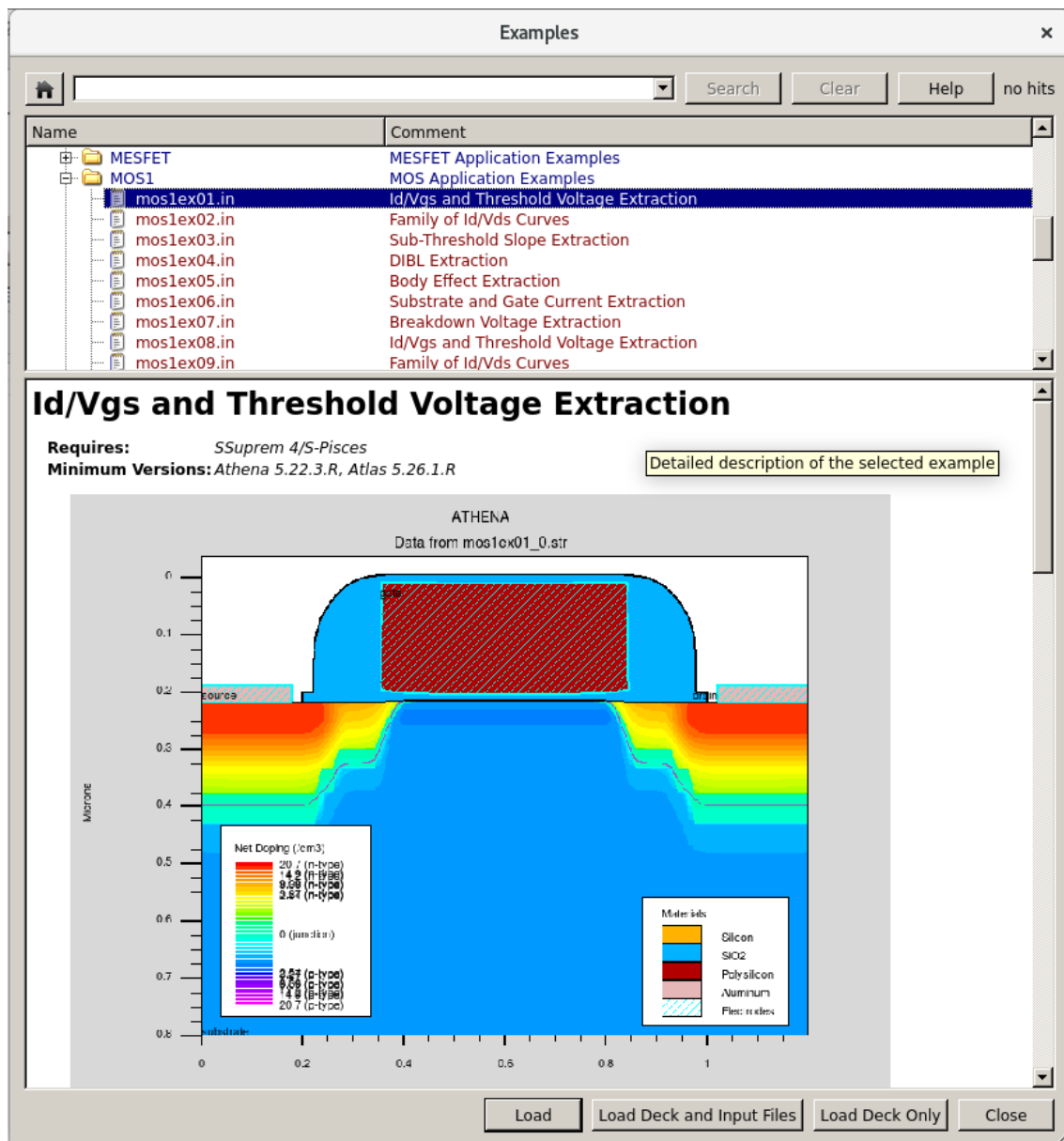


Figure 3-47 Selecting an Example from the Tree

By entering a search string, the database of TCAD examples is searched. All parts of the examples are considered in the search. This includes descriptive text, as well as the deck of the example. Figure 3-48 shows results when searching the examples database for the string **mos1**. The information presented is organized in terms of fields with **section** beginning at the very left and **version** ending at the very right. The first two fields on the left represent the hierarchy of how the TCAD examples are organized. The **section** (here **MOS1**) contains basic MOS examples. The **example** field denotes the name of an example. **title** shows the title of the example. **simulator** shows the simulators used in the example. **technology** shows the technology used in the example. **version** shows the version of DeckBuild used in the example. On the right hand side next to the **Clear** button, the number of hits is indicated. The search returned 16 hits in this example. Hits are automatically ranked such that the best match

always appears at the top. Entering a basic string will consider all fields in the search. You can also opt to refine your search by limiting the search to a particular field only.

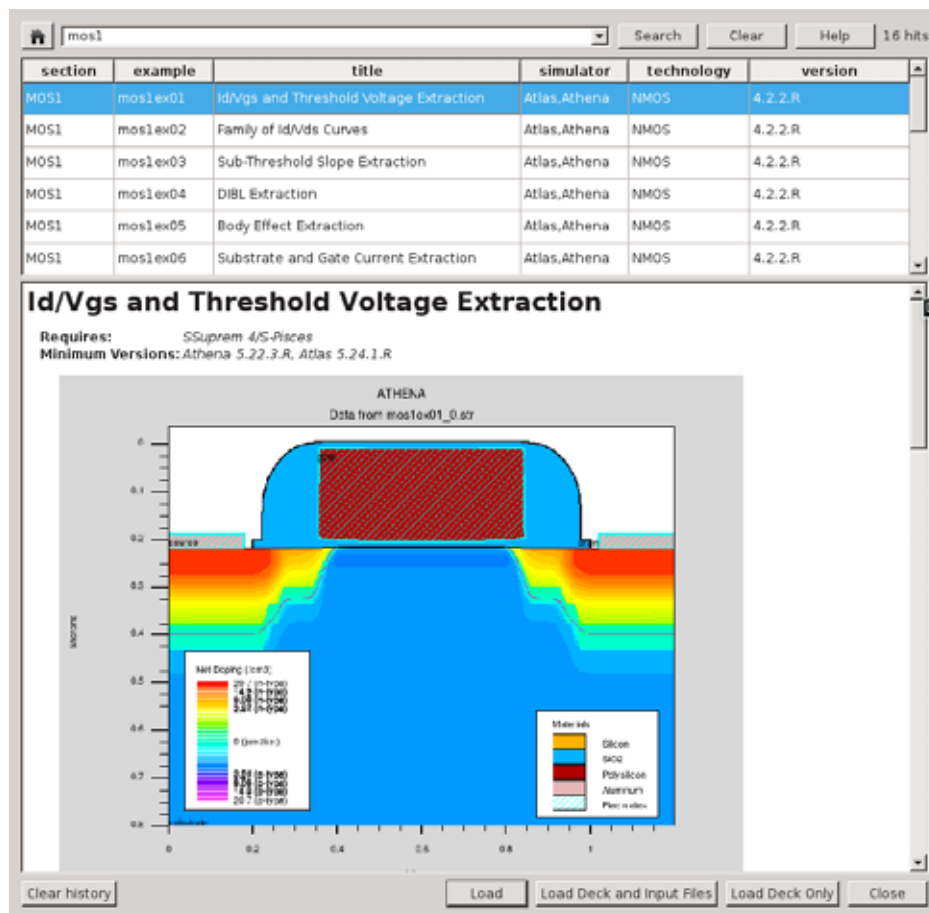


Figure 3-48 Search Results for String mos1

Figure 3-49 shows another search over all fields. This time, the keyword **threshold** is being searched. The search returns 69 hits with an example from the **section SONOS** ranked at the top. Refine search by adding more keywords as shown below.

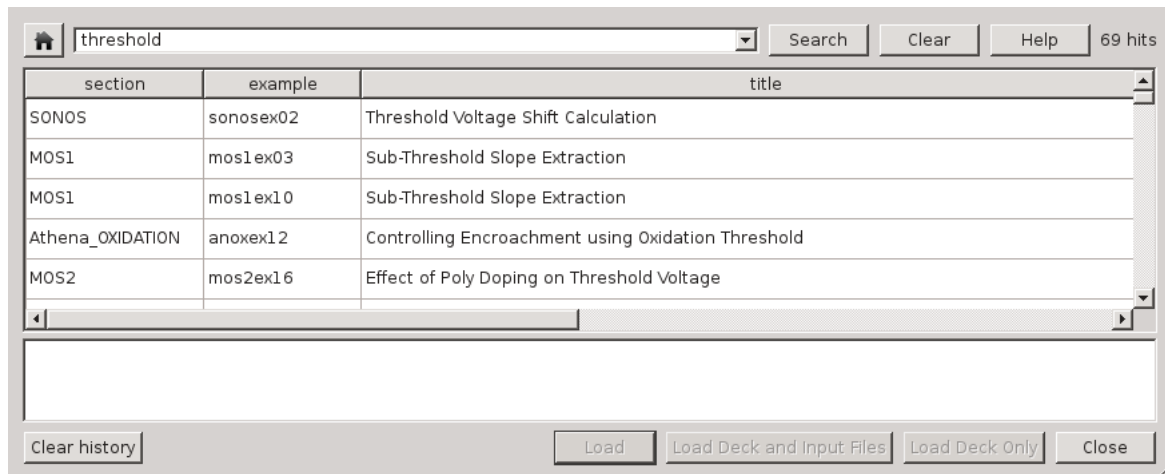


Figure 3-49 Searching for “threshold”

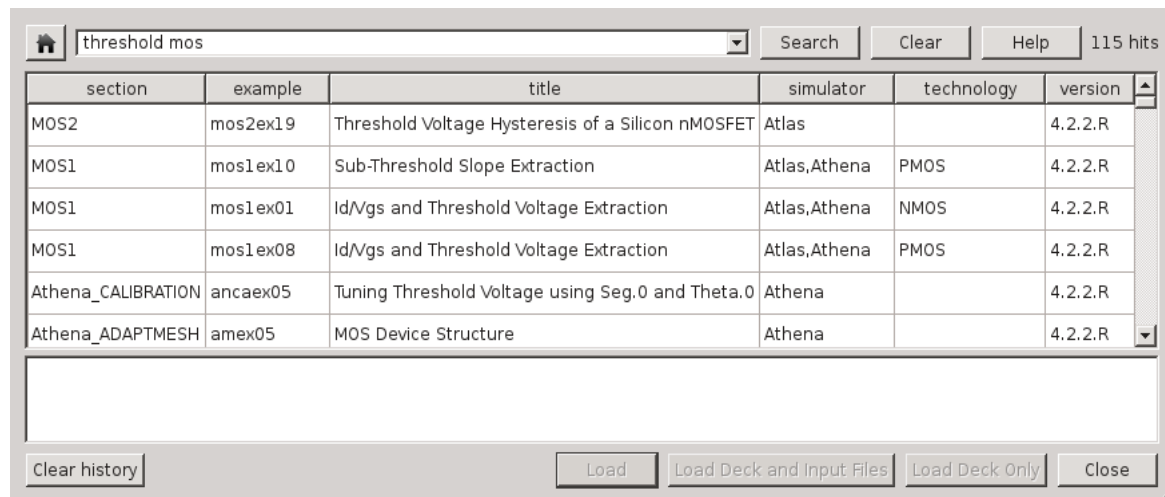


Figure 3-50 Search Expression Consisting of More Than a Single Keyword

Figure 3-50 shows the case where above search was modified to contain threshold mos. The effect of adding a second keyword is that now both keywords are being matched. Every example that matches either threshold or mos is returned with the best matches (both keywords) at the top. A total of 115 hits is returned in this case.

To do a slightly different search that returns only documents containing both keywords, make the keywords mandatory by adding a plus sign (+) before each search term as shown in Figure 3-51. Compared to Figure 3-50, the number of hits was reduced to 71.

Search interface showing results for the query `+threshold +mos`. The search bar contains `+threshold +mos` and the results count is 71 hits. The table below shows the search results.

section	example	title	simulator	technology	version
MOS1	mos1ex05	Body Effect Extraction	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex12	Body Effect Extraction	Atlas,Athena	PMOS	4.2.2.R
TFT	tftex01	Passivated Device	Atlas	Amorphous Silicon TFT	4.2.2.R
TFT	tftex02	Un-Passivated Device	Atlas	Amorphous Silicon TFT	4.2.2.R
TFT	tftex03	Passivated Device	Atlas	Polysilicon TFT	4.2.2.R
TFT	tftex04	Un-Passivated Device	Atlas	Polysilicon TFT	4.2.2.R

Buttons at the bottom: Clear history, Load, Load Deck and Input Files, Load Deck Only, Close.

Figure 3-51 Search with Mandatory Keywords

For results that do not contain a particular keyword, use the minus sign (-) as shown in [Figure 3-52](#). The number of hits is now 44.

Search interface showing results for the query `+threshold -mos`. The search bar contains `+threshold -mos` and the results count is 44 hits. The table below shows the search results.

section	example	title	simulator	technology	version
SONOS	sonosex02	Threshold Voltage Shift Calculation	Atlas		4.2.2.R
MOS2	mos2ex16	Effect of Poly Doping on Threshold Voltage	Atlas		4.2.2.R
FERRO	ferroex02	Threshold Shift in 2D FET	Atlas		4.2.2.R
EPR0M	epmex03	Controlling the Capacitative Coupling	Atlas,Athena		4.2.2.R
LASER	laserex09	Disk MQW Laser	Atlas		4.2.2.R
MOS2	mos2ex19	Threshold Voltage Hysteresis of a Silicon nMOSFET	Atlas		4.2.2.R

Buttons at the bottom: Clear history, Load, Load Deck and Input Files, Load Deck Only, Close.

Figure 3-52 Search with Keywords to Exclude

By selecting one of the examples from the results list, you can view the description of the example ([Figure 3-53](#)).

mos1 Search Clear Help 16 hits

section	example	title	simulator	technology	version
MOS1	mos1ex01	Id/Vgs and Threshold Voltage Extraction	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex02	Family of Id/Vds Curves	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex03	Sub-Threshold Slope Extraction	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex04	DIBL Extraction	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex05	Body Effect Extraction	Atlas,Athena	NMOS	4.2.2.R
MOS1	mos1ex06	Substrate and Gate Current Extraction	Atlas,Athena	NMOS	4.2.2.R

Basic MOS Athena to Atlas interface example simulating an Id/Vgs curve and extracting threshold voltage and other SPICE parameters. No advanced features are used in this example so as to demonstrate simple functionality. This example demonstrates:

- Process simulation of a MOS transistor in Athena
- Process parameter extraction (eg. oxide thicknesses)
- Autointerface between Athena and Atlas
- Ample Id/Vgs curve generation with Vds=0.1V
- Parameter extraction for Vt, linear gain (beta) and mobility rolloff (theta)

The process simulation in SSuprem 4 follows a standard LDD MOS process. The process steps are simplified and default models are used to give a fast runtime. The polysilicon gate is formed by a simple geometrical etch. Before this point the simulation is essentially one dimensional and hence is run in Athena's 1D mode. After the poly etch, the structure converts to 2D.

The grid used in this example is defined quite tightly. However the statement `init ... spac.mult=3` relaxes the mesh in X and Y directions by a factor of three. A more typical mesh for MOS simulation can be obtained by setting `spac.mult=1`.

Using DeckBuild's auto-interface, the process simulation structure will be passed into Atlas automatically. This auto-interface therefore allows global optimization from process simulation to device simulation to SPICE model parameter extraction.

The `extract` statement at the end of the file is used to calculate the oxide thickness at that point. The value returned here may be used as an optimization target for calibration. Refer to the **Interactive Tools of the Virtual Wafer Fab** manual for instruction in the use of the **Optimizer**. This value will be appended to a file in the current working directory called `results.final`. When using the VWF automation tools, `extract` values are logged to the worksheet for RSM modeling. Significant use of `extract` statements is strongly recommended, especially at points where in-line Fab measurements are taken.

Electrodes are defined at the end of the process simulation. Metal is deposited and patterned. Then `electrode` statements are used to define the metal regions plus the polysilicon as electrodes for use in Atlas.

In Atlas the first task is to define the models and material parameters for the simulation. The `contact` statement is used to define the workfunction of the gate electrodes, while the `interface` statement defines the fixed charge at the silicon/oxide interface. For simple MOS simulation the parameters CVT and SRH define the recommended models. CVT sets a general purpose mobility model including concentration, temperature, parallel field and transverse field dependence. For more

Clear history Load Load Deck and Input Files Load Deck Only Close

Figure 3-53 Viewing Description of an Example

3.18 Cross-referencing runtime output and the deck

Deckbuild allows you to easily identify what line in a deck has produced which runtime output. You can either right-click anywhere in the deck and choose “**Show in RTO**”, or you can right-click a line of runtime output in the runtime output pane and select “**Show deck line**”.

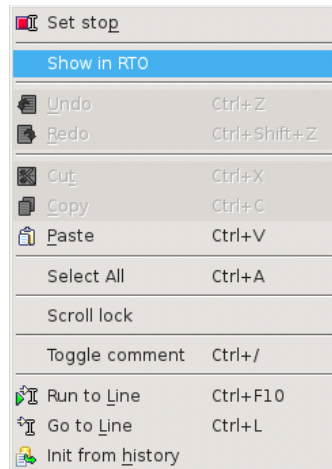


Figure 3-54 Cross-referencing deck and RTO from the deck pane

Figure 3-54 shows the menu entry that is displayed when right-clicking in the deck pane. Figure 3-55 shows the corresponding menu entry when right-clicking in the RTO pane. Figure 3-56 shows how the deck line and the corresponding portion of runtime output are highlighted. Please note, that the highlighting is removed once you move the cursor to a different line in the deck.

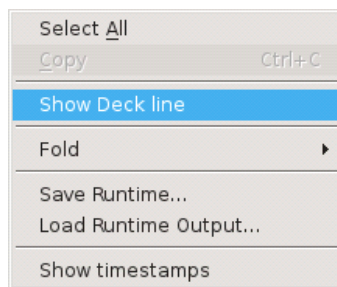


Figure 3-55 Cross-referencing deck and RTO from the RTO pane

The screenshot shows the DeckBuild software interface. The top menu bar includes File, Edit, View, Run, Tools, Commands, and Help. Below the menu is a toolbar with various icons for file operations and simulation control. The main window is divided into two panes. The top pane, labeled 'Deck', contains the following text:

```
line y loc=0.8 spac=0.15
#
init orientation=100 c.phos=1e14 space.mul=2
#pwell formation including masking off of the nwell
#
diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
#
etch oxide thick=0.02
#
#P-well Implant
```

The bottom pane shows the runtime output (RTO) for the command `diffus time=30 temp=1000 dryo2 press=1.00 hcl=3`. The output is a table of solving times:

Solving time (hh:mm:ss.t)	+	[sec]	[%]	[np]
00:00:00.0	+	[1e-05]	[100 %]	[np 100]
00:00:00.0	+	[0.0009]	[9900 %]	[np 100]
00:00:00.0	+	[0.0125]	[1263 %]	[np 100]
00:00:00.0	+	[0.0125]	[100 %]	[np 100]
00:00:00.0	+	[0.3447]	[2758 %]	[np 100]
00:00:00.3	+	[1.248]	[361.9 %]	[np 100]
00:00:01.6	+	[6.542]	[524.4 %]	[np 100]
00:00:08.1	+	[47.25]	[722.2 %]	[np 100]
00:00:55.4	+	[137.1]	[290.1 %]	[np 100]
00:03:12.4	+	[317]	[231.3 %]	[np 98]
00:08:29.5	+	[450]	[142 %]	[np 96]
00:15:59.5	+	[450]	[100 %]	[np 94]
00:23:29.5	+	[390.5]	[86.78 %]	[np 92]
00:30:00.0				

Below the RTO table, the user has entered the following commands:

```
ATHENA> #
ATHENA> etch oxide thick=0.02
ATHENA> #
ATHENA> #P-well Implant
```

At the bottom of the window, there is an 'Output' section with a 'Scroll to bottom' checkbox and a 'Clear' button. The status bar at the very bottom shows: 'Line: 13 Column: 2 | Finished executing - | Size of generated files : 135 | Free space : 125.1 GB | DeckBuild Copyright © silvaco'.

Figure 3-56 Cross-referencing: deck line and RTO portion are highlighted

3.19 Folding runtime output

Deckbuild allows you to reduce the amount of runtime output to only display the first line that is returned by a command. [Figure 3-57](#) and [Figure 3-58](#) show a folded and an unfolded portion of runtime output. Please note the changed symbol at the left area of the RTO pane, which changes from '-' to '+' when the text is collapsed and vice versa when it is uncollapsed. You can also fold/unfold the whole RTO pane in one go by using one of the two entries of the 'Fold' submenu as shown in [Figure 3-59](#).

```

ATHENA> #
ATHENA> diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
Solving time (hh:mm:ss.t) 00:00:00.0 + [1e-05 sec] [100 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:00.0 + [0.0009 sec] [9900 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:00.0 + [0.0125 sec] [1263 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:00.0 + [0.0125 sec] [100 %] [np 100] *
Solving time (hh:mm:ss.t) 00:00:00.0 + [0.3447 sec] [2758 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:00.3 + [1.248 sec] [361.9 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:01.6 + [6.542 sec] [524.4 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:08.1 + [47.25 sec] [722.2 %] [np 100]
Solving time (hh:mm:ss.t) 00:00:55.4 + [137.1 sec] [290.1 %] [np 100]
Solving time (hh:mm:ss.t) 00:03:12.4 + [317 sec] [231.3 %] [np 98]
Solving time (hh:mm:ss.t) 00:08:29.5 + [450 sec] [142 %] [np 96]
Solving time (hh:mm:ss.t) 00:15:59.5 + [450 sec] [100 %] [np 94]
Solving time (hh:mm:ss.t) 00:23:29.5 + [390.5 sec] [86.78 %] [np 92]
Solving time (hh:mm:ss.t) 00:30:00.0
ATHENA> #

```

Figure 3-57 Unfolded runtime output

```

ATHENA> #
+ ATHENA> diffus time=30 temp=1000 dryo2 press=1.00 hcl=3
ATHENA> #

```

Figure 3-58 Folded runtime output

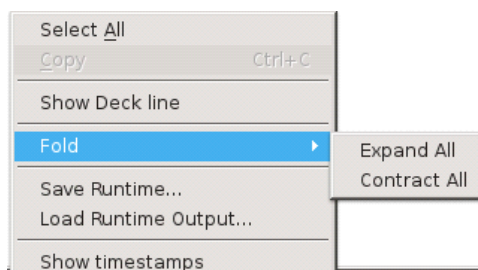


Figure 3-59 Folding text: fold/unfold all of the runtime output

3.20 Visualizing VictoryProcess line statements

Deckbuild offers simulator specific help for the `LINE` and `CARTESIAN` commands of VictoryProcess.

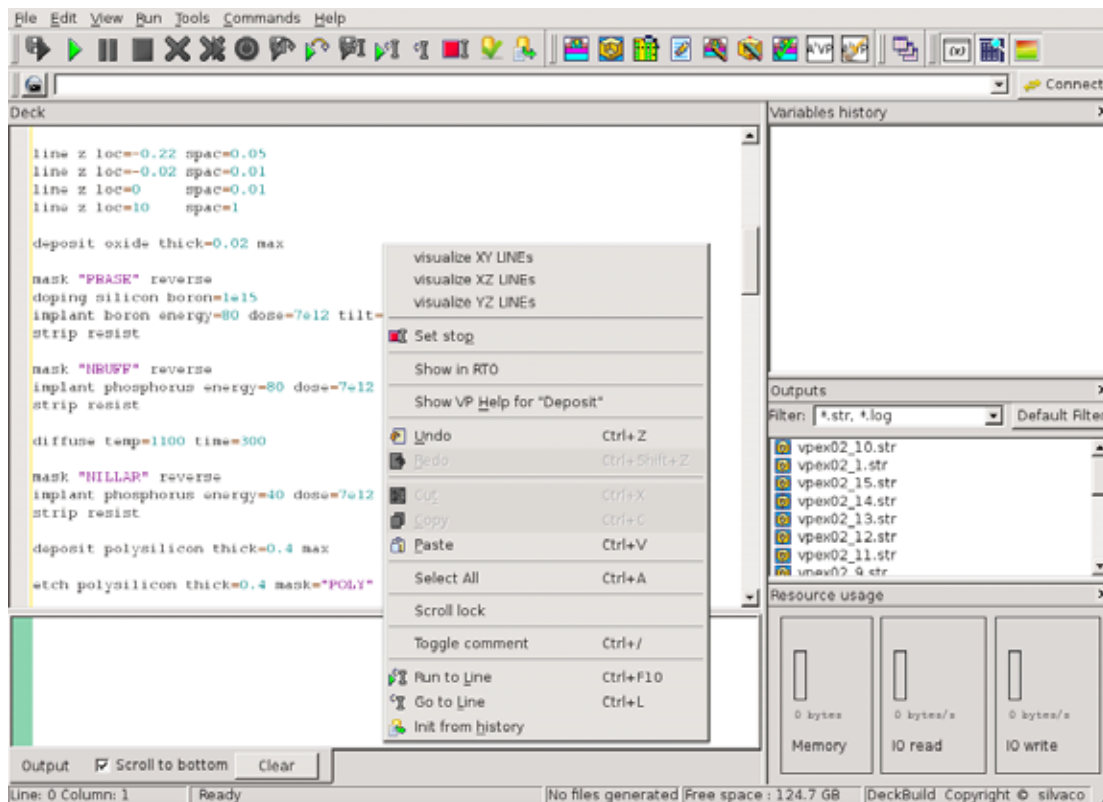


Figure 3-60 Visualizing VictoryProcess LINE statements from the context menu

Figure 3-60 displays the context menu that opens when right-clicking in a VictoryProcess deck, which uses `LINE` and `CARTESIAN` statements. Depending on what `LINE` statements (`x`, `y`, or `z`), up to three options conforming to the three planes, `xy`, `xz`, `yz` are offered. Note, that only lines from the top of the deck up to the position where you right-click are taken into account.

Figure 3-61 shows the visualization of the `XY` plane.

Please note, that for a `LINE` (or `CARTESIAN`) statement to be taken into account it must be free of variables. Statements of the form:

```
set loc1=-0.22
line z loc=$loc1 spac=0.05
```

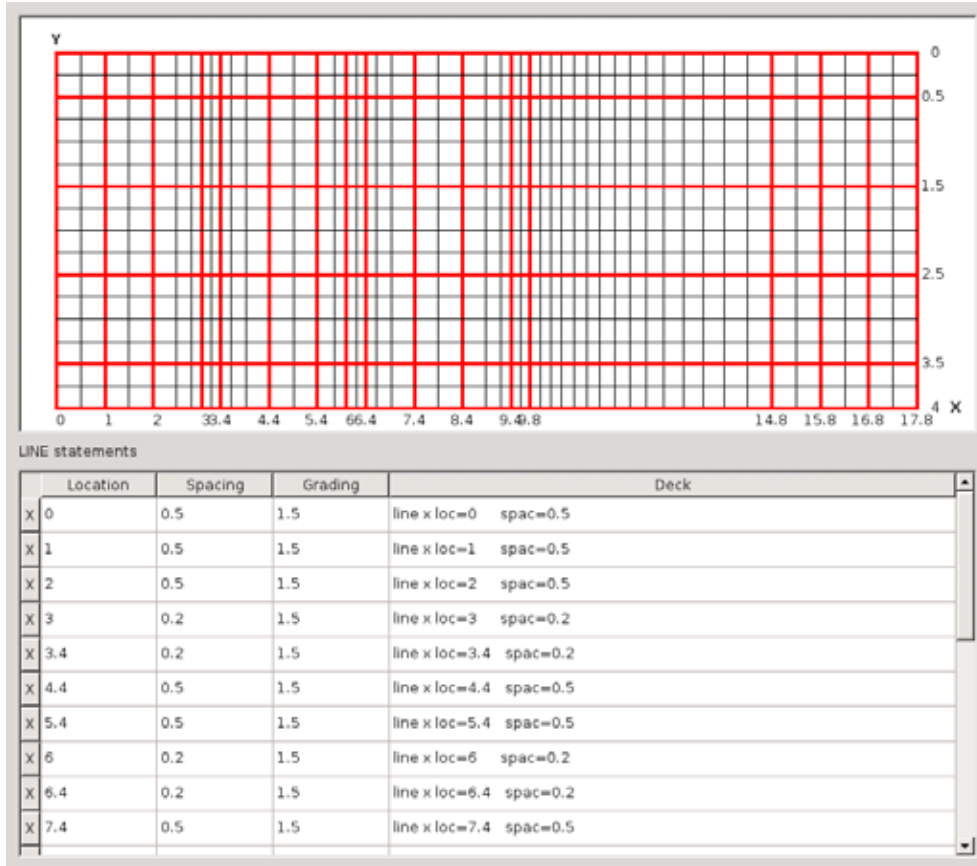


Figure 3-61 Visualization of VictoryProcess LINEs in the XY plane

will not be supported and Deckbuild will ignore this particular LINE (or CARTESIAN) statement.

3.21 Context sensitive help system

Deckbuild offers several ways to get help. By selecting the “**Help→Deckbuild Help**” menu entry (or by pressing the F1 key), the deckbuild PDF manual which you are reading right now will open. (see also [Section 3.14 Edit Menu](#))

Deckbuild also offers help directly on the deck syntax without the need to open the simulator manual. You can right-click on any command in the deck. This is supported for the simulators VictoryProcess, VictoryMesh and for the Deckbuild internal commands. [Figure 3-62](#) shows the menu that is populated then right-clicking anywhere in the VictoryProcess deck. In this case the command 'save' was clicked and the menu entry allows to open the help on the same command (Show VP Help for “Save”)

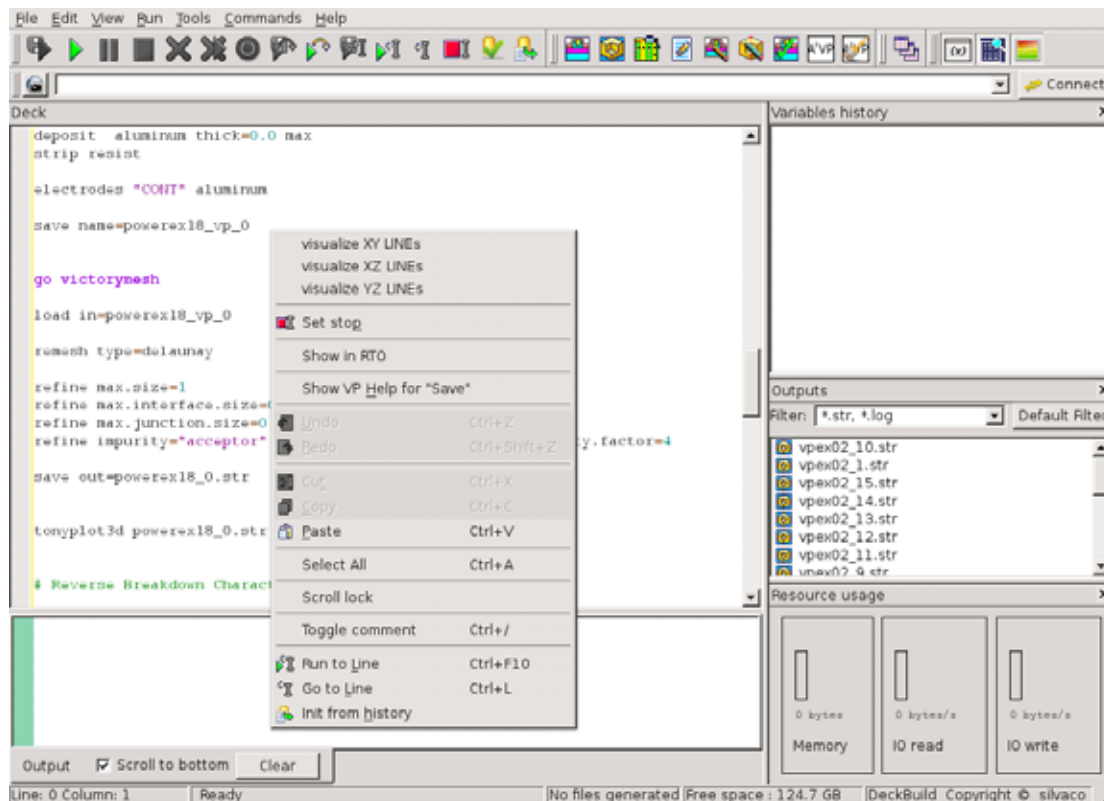


Figure 3-62 VictoryProcess Help menu shown on right-clicking into a VictoryProcess section

[Figure 3-63](#) shows the pop-up that opens when the Show VP Help for “Save” action is chosen.

```

SAVE

This statement saves the status of the simulation (structure) to the file system. The
simulation status consists of five files:
.) the geometry file
.) the volume data file
.) the mask file
.) the electrode file (in process mode only)
.) the properties file
.) the file with postponed commands

The geometry file contains the full geometrical representation of the structure, including
geometrical and topological information. It can be identified
.) by the file extension ".svf" (when the simulation is running in process mode), or
.) by the file extension ".str" (when the simulation is running in cell mode)

The volume data file contains the full volume data representation of the structure, including
all volumetric data. It can be identified by the file extension ".dop".

The mask file contains a file representation of the internal mask set of Victory Process,
including all masks and all electrodes. It can be identified by the file extension ".lay".

The electrode file contains information about how electrodes have been assigned to the
structure. This is necessary when creating visual representations. It can be identified
by the file extension ".ele".

The properties file contains all material and simulation properties that have been
assigned in the command file to override data from the material database.
It can be identified by the file extension ".prp".

The postponed commands file contains the requests, which were not processed yet. For example,
the manual refinement queue. It can be identified by the file extension ".ppc".

All of the above mentioned files will have the same file base-name when saving the simulation
status. Thereby, all files composing a full simulation status can be easily identified.

Syntax

SAVE NAME=<Name>
[ ASCII ]

Parameter      Type      Default      Units
NAME           Character
ASCII          Logical    OFF/FALSE

Description

```

Figure 3-63 VictoryProcess help for the save command

Figure 3-64 shows what happens when you right-click into a VictoryMesh deck. You can see that the menu entry has slightly changed and now reads:

“Show VictoryMesh help for refine”

By choosing this action you will open the window as shown in Figure 3-65. The window allows you to also navigate among the various other VictoryMesh commands that are available.

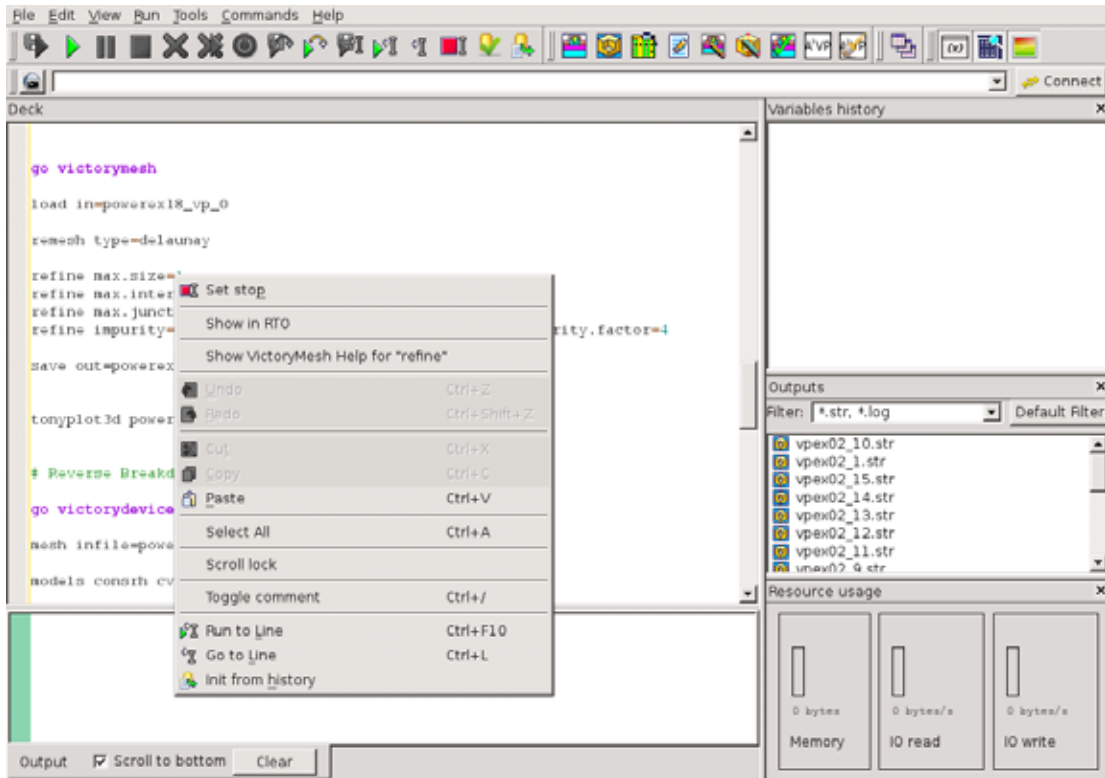


Figure 3-64 VictoryMesh Help menu shown on right-clicking into a VictoryMesh section

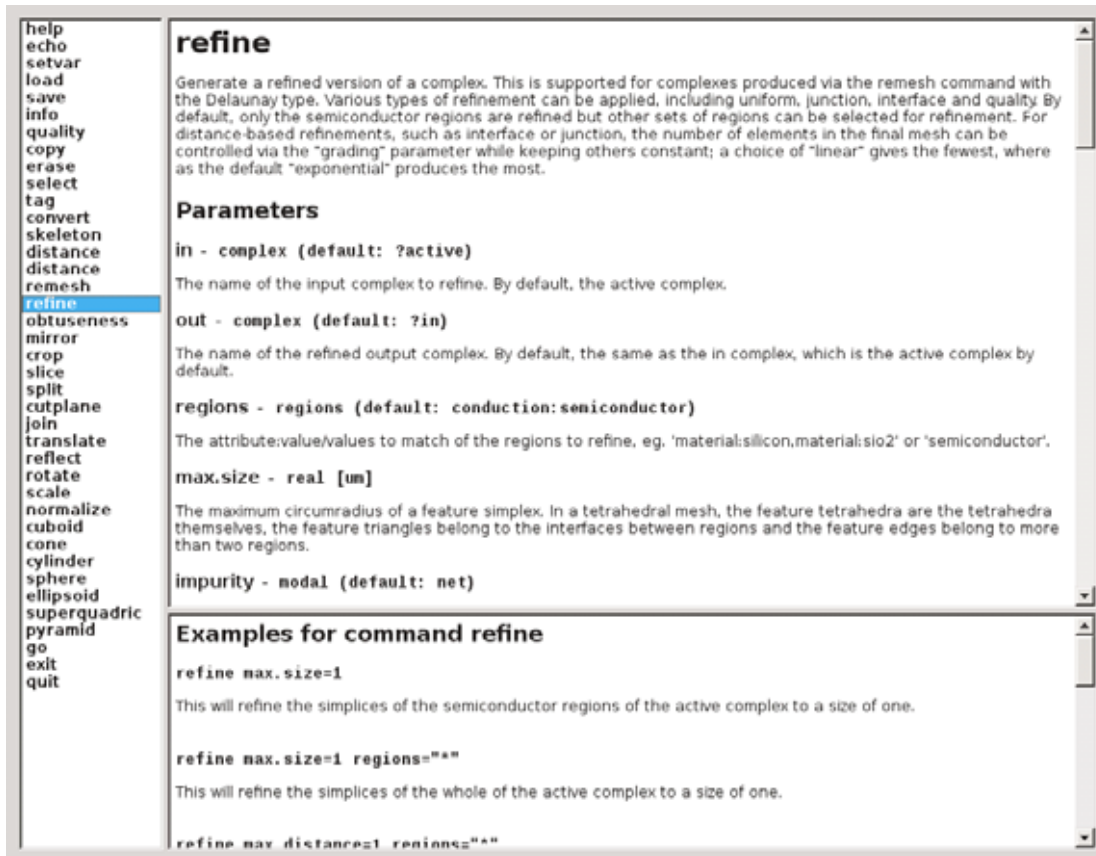


Figure 3-65 VictoryMesh help for the `refine` command

Finally, [Figure 3-66](#) shows the menu entry when you right-click any of the internal deckbuild commands. In this example a 'set' command was chosen. The menu entry thus reads:

“Show Deckbuild Help for set”

If you select the menu entry then the window shown in [Figure 3-67](#) opens. As with the VictoryMesh help the window allows you to navigate around various other commands, like IF, LOOP, SYSTEM and others.

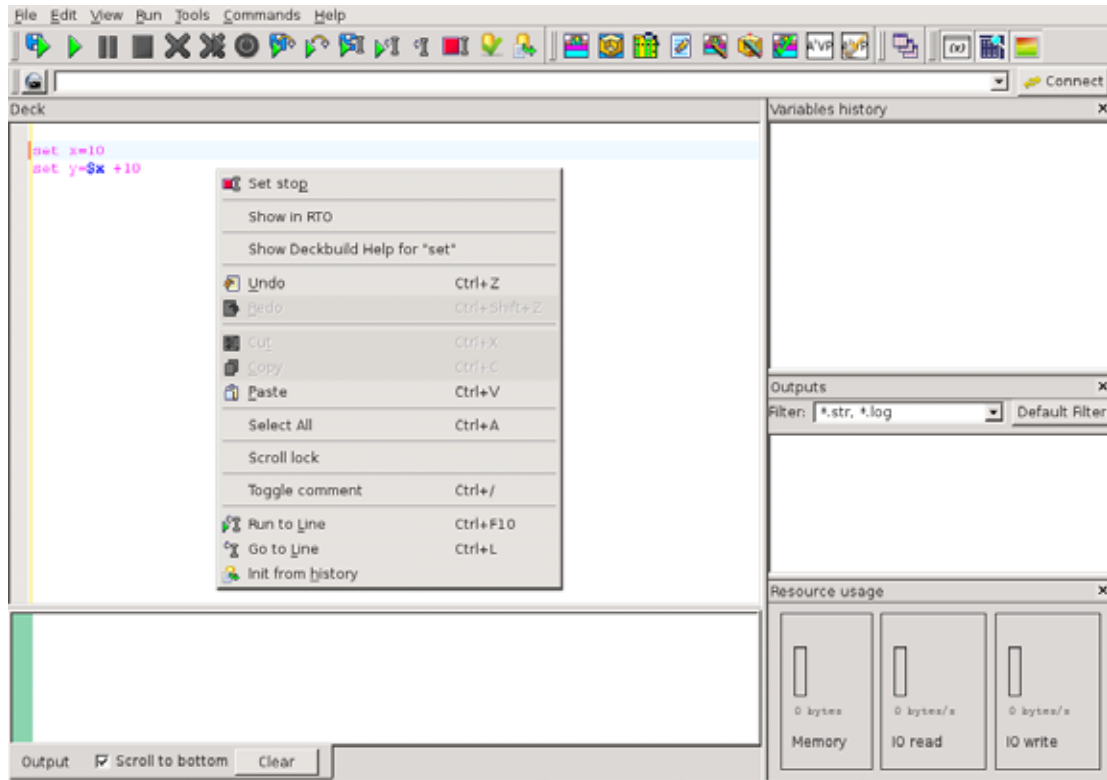


Figure 3-66 Deckbuild help menu shown when right-clicking on a Deckbuild command



set

This statement assigns strings or numbers to variables

SET variablename=assignment
where:

- <assignment>: Placeholder for any string or number value, which will be assigned to the variable defined by the placeholder <variablename>
- <variablename>: Placeholder for any variable name that can be used later within the command file.

This statement assigns strings or numbers to variables, which can be used later on in the command file. The SET statement defines a variable and assigns a value to this variable. The content of the variable can be accessed later in the command file by means of the '\$' prefix. You can also perform numerical operations on numbers or numerical variables using the SET statement. You can find full documentation of the SET statement in the DeckBuild User's Manual.

For more detailed description of these statements (e.g. nested if-else statements, or possible structure of the) see the DeckBuild manual (Section "Statements").

Examples for command set

```
SET MYDOSE=1e13  
Set a variable to a numerical value
```

```
SET MYMASK=Gate  
Set a variable to a string value
```

```
SET HAI MYDOSE = MYDOSE / 2
```

Figure 3-67 Deckbuild help for the `set` command

3.22 Commands

3.22.1 Deck Writing Paradigm

Generally, DeckBuild supports several ways of writing an input deck: You can copy/paste portions from another deck, or you can write deck line by line, or you can use popups to create statements for a simulator. You can mix and match and use each as appropriate. Process simulation, for example, is an inherently sequential operation. The same basic commands (implant, diffuse, etch, and deposit) are used over and over again. Victory Process and Athena are good examples of how this paradigm works, because each popup has a button used to just write the syntax for that popup/command.

3.22.2 Commands Menu

The **Commands** menu is the primary means of accessing dialogs used to write the input deck. Typically, each item on the menu is associated with a dialog that contains controls used to specify an input deck command. For instance, invoking Implant causes the Athena Implant dialog to appear. Because Victory Process is compatible with Athena, you can use the same Implant dialog to define an Implant command for Victory Process.

3.22.3 Parsing the Deck

DeckBuild has a built-in feature that allows parsing any part of a deck to automatically configure the appropriate dialogs. For example, to repeat a previous implant process command with some minor changes, invoke the parser on the next `IMPLANT` statement. Then, apply the needed changes, place the text cursor in the proper location, and press the **WRITE** button.

[Figure 3-68](#) illustrates how the parser is used. Select portions of a line or the lines as a whole (double-click on a line to select it a a whole) and right-click. A menu will open, allowing you to open the identified command dialog. In this example, the dialog for the implant command will be opened.

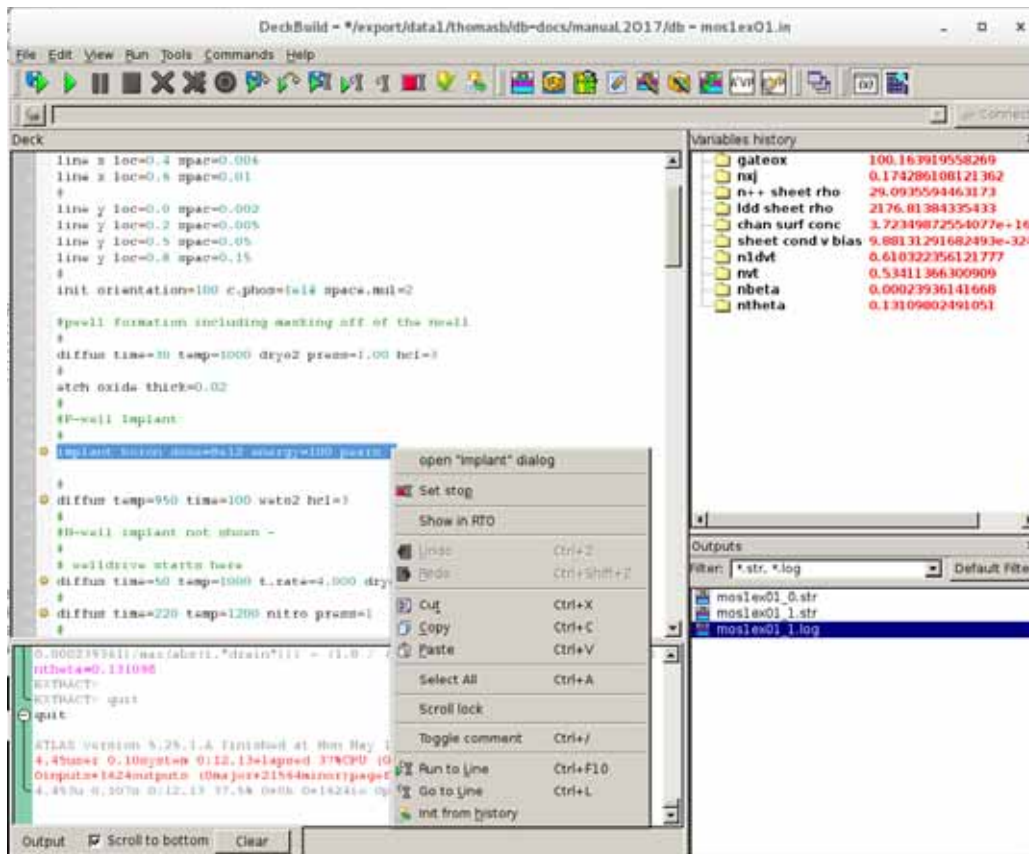


Figure 3-68 Parsing a Line of Deck

Note: Parse Deck does not carry over parameters into the dialog that are not specified. For example, if you are parsing the line `implant boron`, the values of energy and dose will not be altered from whatever previous value they had on the dialog.

3.22.4 Process Simulators

Figure 3-69 shows how the Process commands menu, consisting of dialogs for Implant, Diffus, Deposit, and others. Some menus do have submenus in variants of a command are available. Figure , shows this for the **Etch** command menu, which has two sub-menu entries called Etch and Rate Etch .

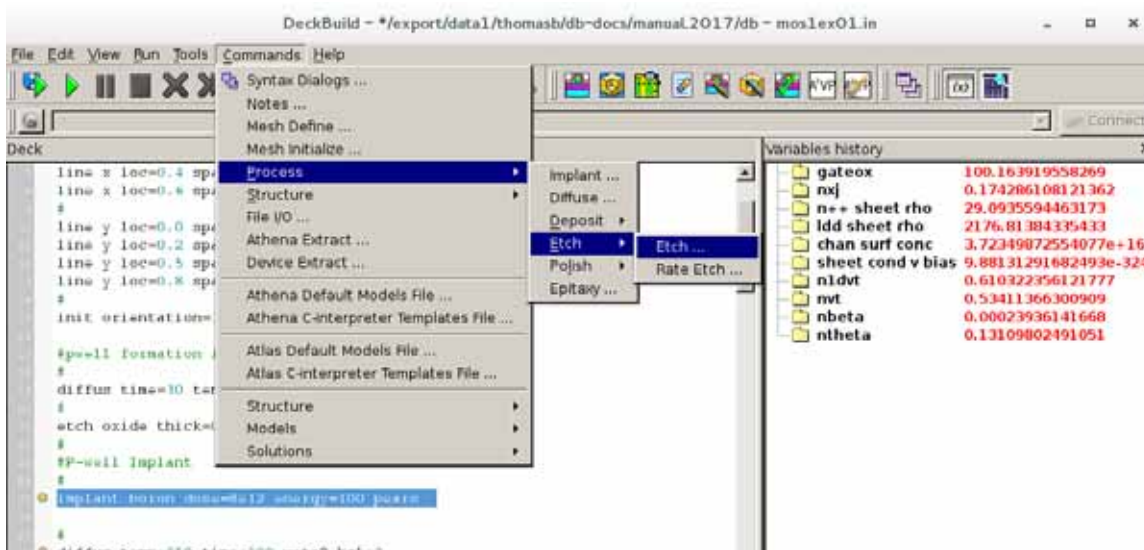


Figure 3-69 Invoking Process Etch Command

3.22.5 Writing a Process Input Deck

Since process fabrication is itself an inherently sequential operation, simply choose the command of interest from the **Commands** menu. A corresponding dialog appears that has controls laid out to represent the variable parameters available for the command. For example, [Figures 3-70, 3-71, 3-72, and 3-73](#) show the Diffusion dialog. The dialog is organized as a tabbed view presenting four tabs: **Time/Temp**, **Ambient**, **Impurities**, and **Models**. These correspond to the various parameters of the diffuse command.

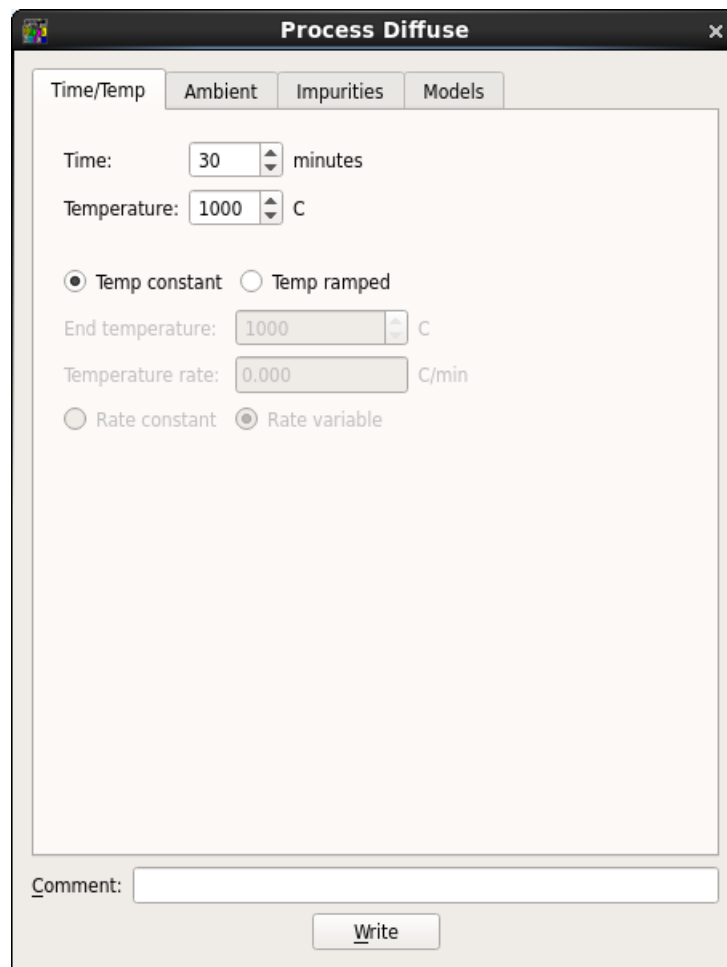
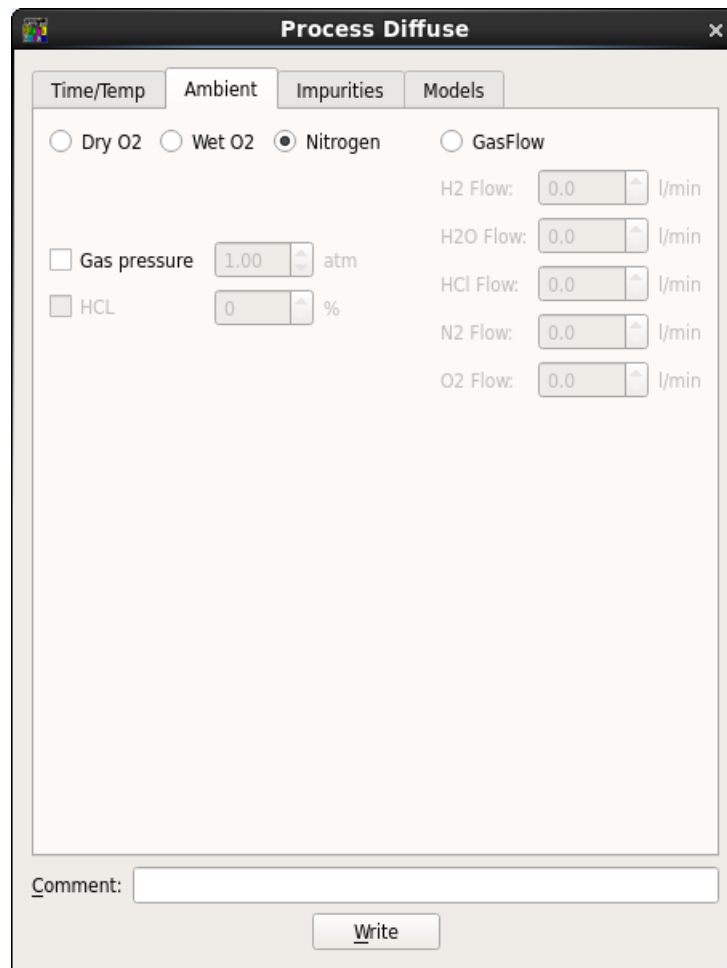


Figure 3-70 Diffuse Dialog–Time/Temp Parameters



The screenshot shows the 'Process Diffuse' dialog box with the 'Ambient' tab selected. The dialog has four tabs: 'Time/Temp', 'Ambient', 'Impurities', and 'Models'. Under the 'Ambient' tab, there are four radio buttons: 'Dry O2', 'Wet O2', 'Nitrogen' (which is selected), and 'GasFlow'. Below these are two checkboxes: 'Gas pressure' (unchecked) and 'HCL' (checked). The 'Gas pressure' checkbox is followed by a numeric input field containing '1.00' and the unit 'atm'. The 'HCL' checkbox is followed by a numeric input field containing '0' and the unit '%'. To the right of these are five flow rate controls, each consisting of a label, a numeric input field, and a unit: 'H2 Flow: 0.0 l/min', 'H2O Flow: 0.0 l/min', 'HCl Flow: 0.0 l/min', 'N2 Flow: 0.0 l/min', and 'O2 Flow: 0.0 l/min'. At the bottom of the dialog, there is a 'Comment:' label followed by a text input field and a 'Write' button.

Figure 3-71 Diffuse Dialog–Ambient Definition

When you adjust all the controls to reflect the process step to be performed, click the **WRITE** button. A line (or sometimes several lines) of text is written to the deck at the location of the text cursor. If desired, verify the cursor's location before clicking **WRITE**, although DeckBuild automatically detects if the caret is in the middle of a line and moves it if necessary. Build the entire process deck by invoking the dialogs as needed from the **Commands** menu, setting the controls, and writing the deck one dialog at a time. You can also parse the deck. That is, read a line or lines of syntax from the deck and automatically configure the correct dialogs.

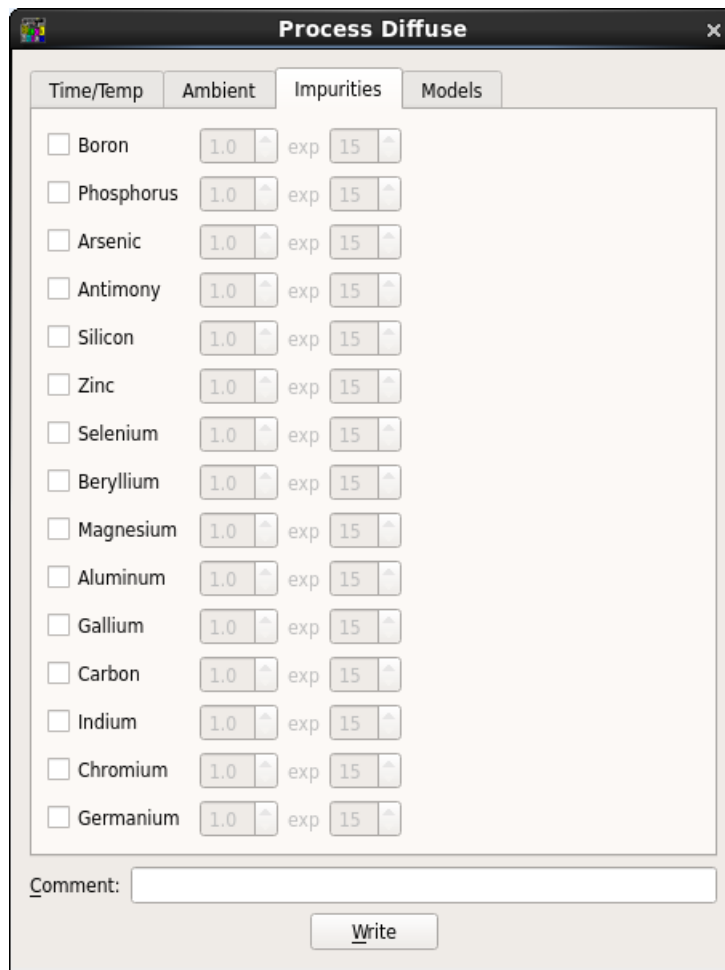


Figure 3-72 Diffuse Dialog-Impurities

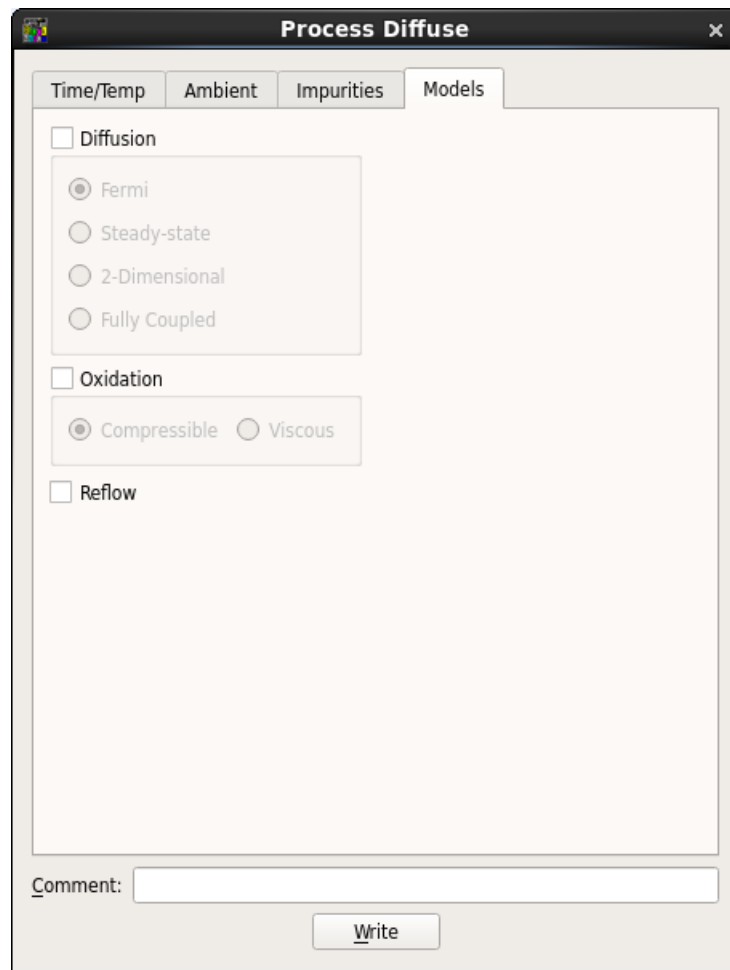


Figure 3-73 Diffuse Dialog-Models

3.23 Preferences

Many settings of DeckBuild are customizable. To do this, open the Preferences panel by selecting **Edit**→**Preferences** (see [Figure 3-74](#)).

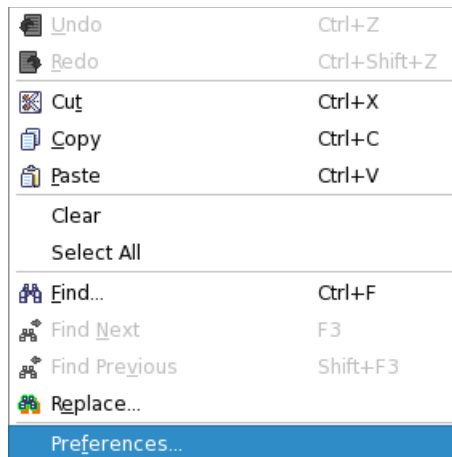


Figure 3-74 Opening Preferences

[Figure 3-75](#) shows the Main Preferences dialog. On the left-hand side, you can see the various areas ranging from **Manage Preferences** to **Simulation Settings**. The right-hand side of the preferences dialog shows the settings that are available within a particular area. This part will change everytime you select a different area. In [Figure 3-75](#), **Manage Preferences** area is selected.

The **Manage Preferences** area offers the four buttons:

- **Import** – Imports settings from a previous export.
- **Export** – Exports the current preferences settings into a file.
- **Factory Settings** – Resets all preferences to their factory defaults.
- **Recent Files** – Clears the recent files lists in the editor.

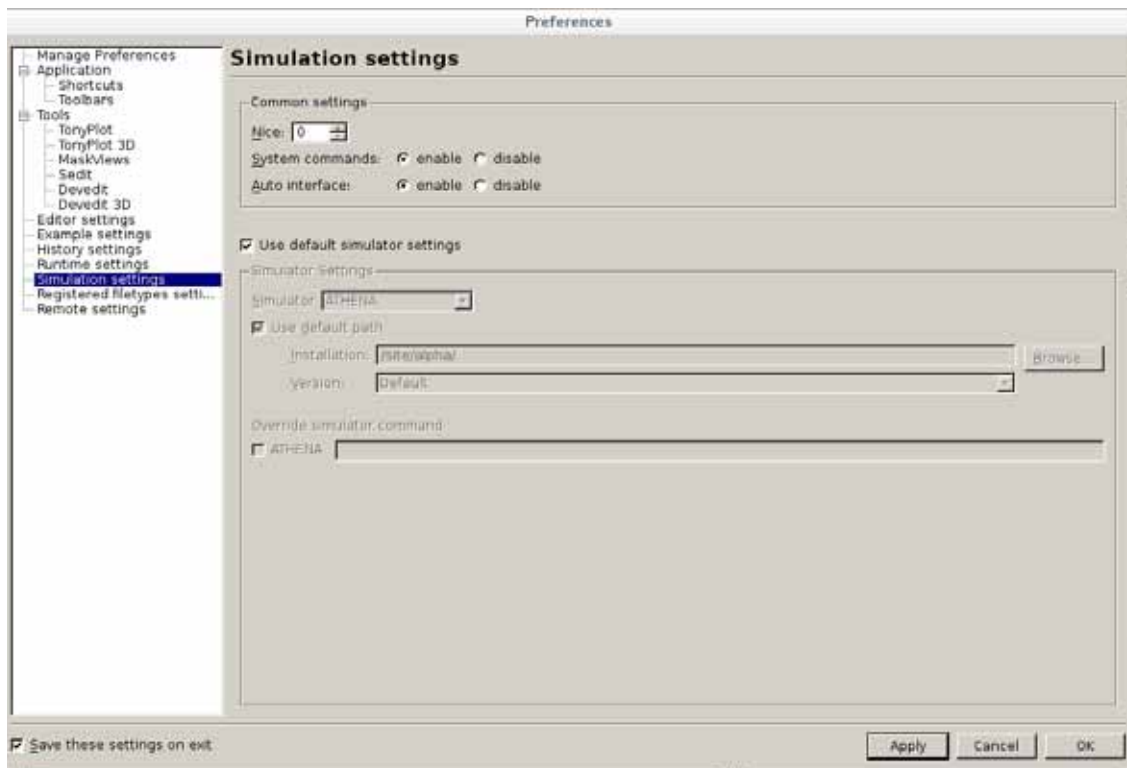


Figure 3-75 Preferences Panel

When you select **Import** or **Export**, a file selection dialog will appear (see [Figure 3-76](#)). Select a file and press the **Import** button to save the preferences to it.

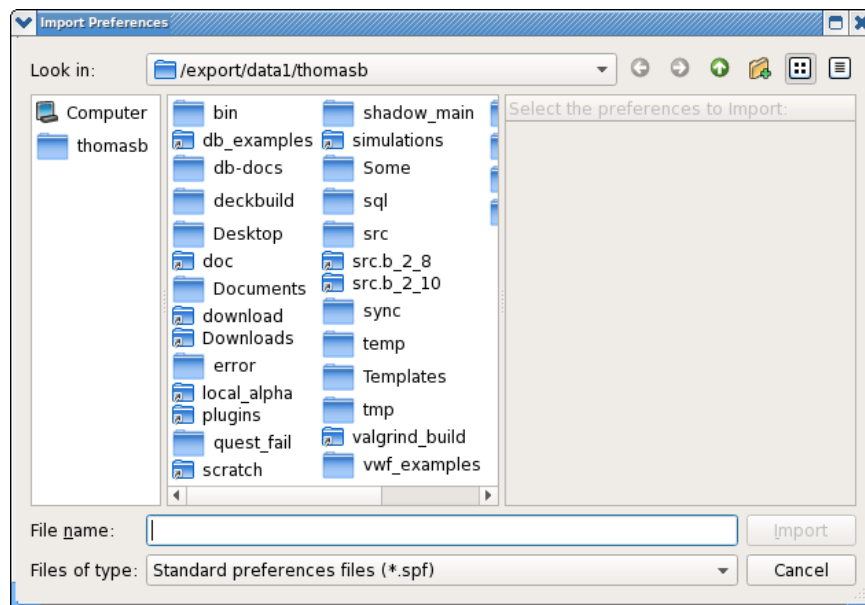


Figure 3-76 File Selection Dialog

[Figures 3-77](#) and [3-78](#) show the confirmation dialogs that appear if you select either **Factory Settings** or **Recent Files**.

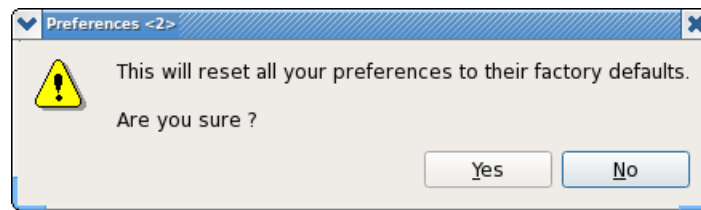


Figure 3-77 Confirmation Dialog: reset to factory defaults

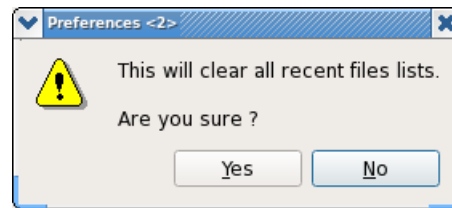


Figure 3-78 Confirmation Dialog: clear recent files list.

3.24 Application

This area allows you to define settings related to the application shortcuts and toolbars. [Figure 3-79](#) shows the settings for shortcuts, whereas [Figure 3-80](#) shows the **Toolbars** settings. In addition to the **View** menu ([Section 3.4.1 The View Menu](#)), the **Toolbars** settings allow you to also define the icon size and whether hints are displayed or buttons are underlined with a textual description.

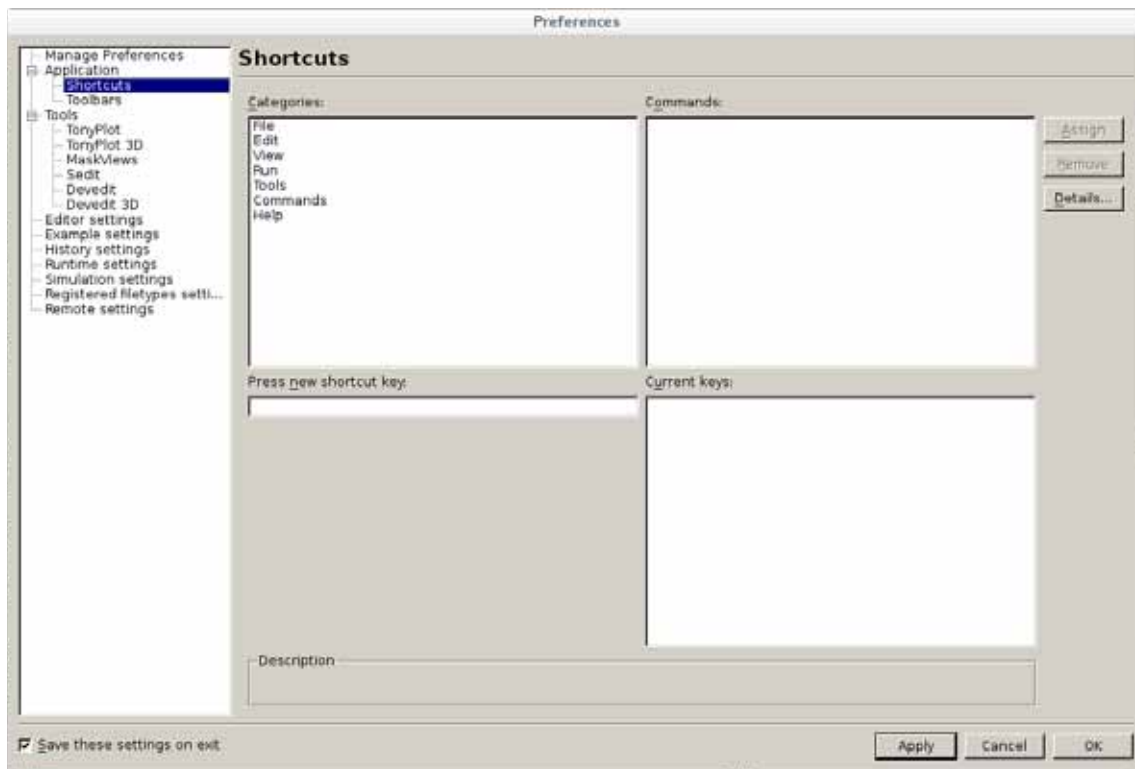


Figure 3-79 Configuring Application Shortcuts

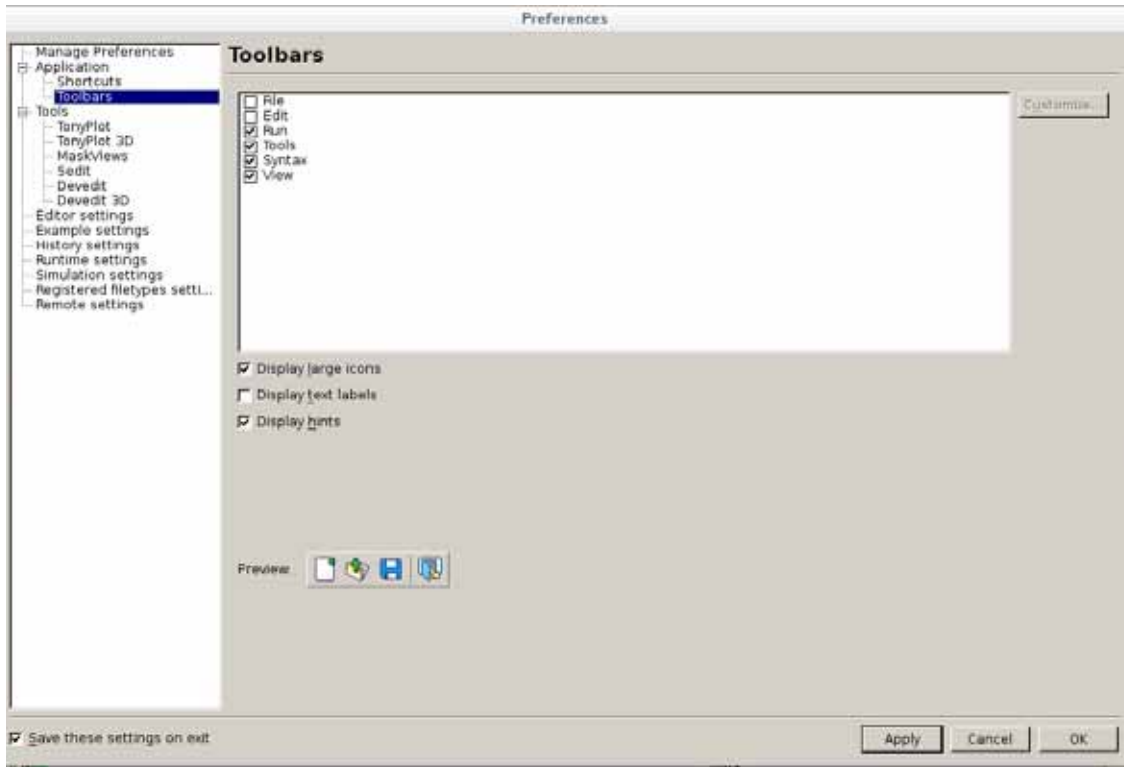


Figure 3-80 Configuring Toolbars

3.25 Tools

The **Tools** area is split into five sections corresponding to TonyPlot, TonyPlot 3D, MaskViews, Sedit, and Devedit. Each selection allows you to choose the installation location and version. Normally, you are not concerned with the installation location as this is where DeckBuild started. However, if you have several installation trees (e.g., to test a new package in a different location first), you can also pass options to a tool. For example, when connected over the network, it may make sense to use the `-nohw` option to the TonyPlot 3D tool to avoid using hardware acceleration. [Figure 3-81](#) shows settings for the TonyPlot tool.

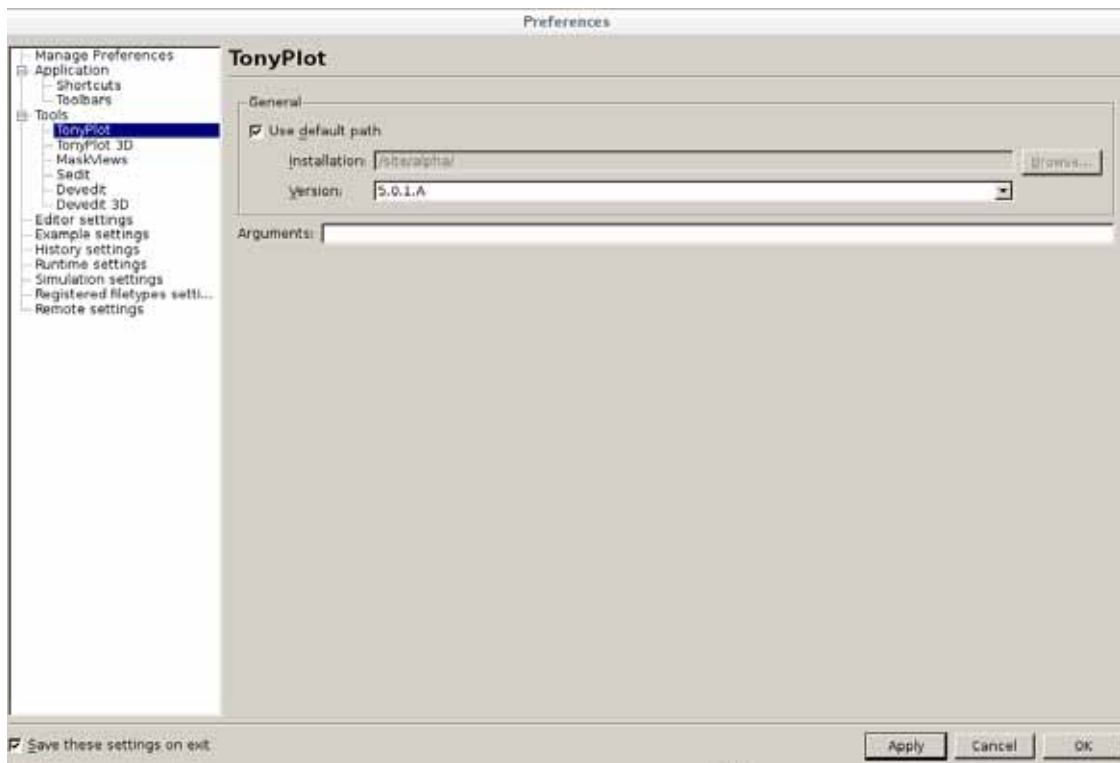


Figure 3-81 Settings for TonyPlot

3.26 Editor Settings

Figure 3-82 shows the preferences settings for the Editor.

With the first group of settings at the very top, you can define the color and font for each existing style used by the Editor.

To do this you select items in Category and Style lists in left area. Then you can set the color and font for this style in the right area.

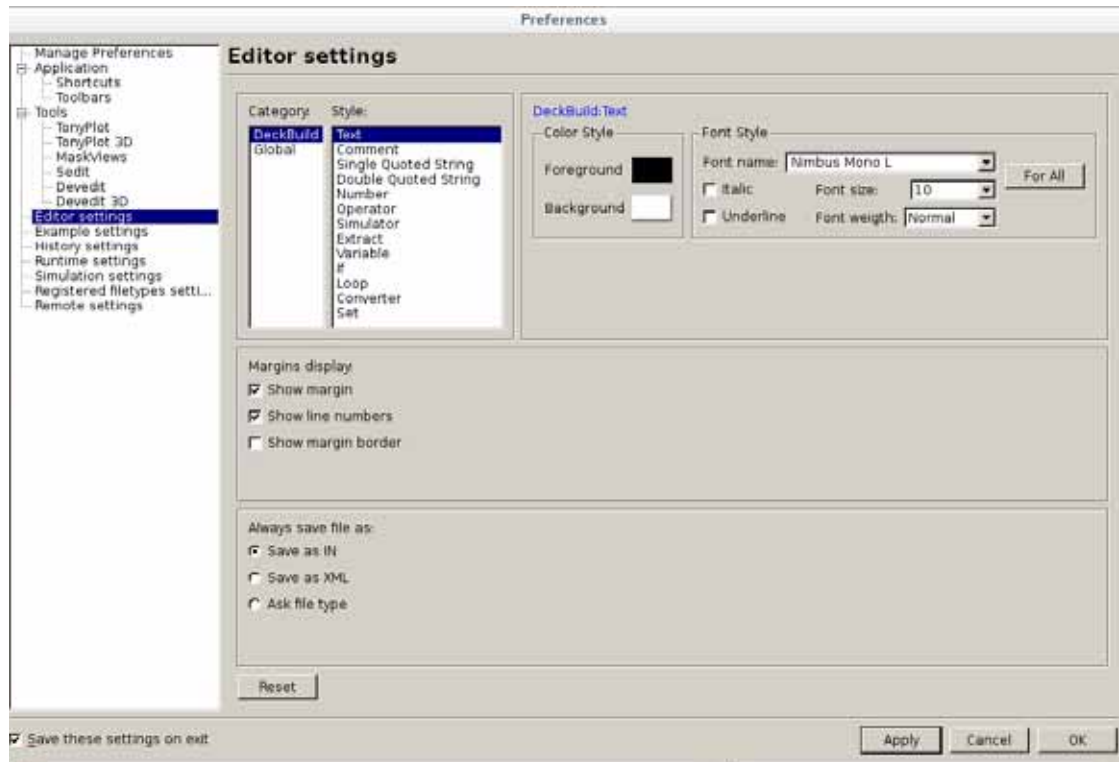


Figure 3-82 Editor Settings

The second group allows you to turn the margin and line number display on and off. This is also available in the View menu ([Section 3.4.1 The View Menu](#)).

The last group of settings allows you to decide which file format to use. The default file format is .IN, which is a plain text format and is also readable by previous DeckBuild versions. This version of DeckBuild also supports the use of an XML format, which basically allows to keep extra information (e.g., defined stops or optimizer experiments). The XML file cannot be opened by previous DeckBuild versions. Note that you can always open an XML file and save it as normal .IN file. This allows backwards compatibility.

3.27 History and File Settings

Figure 3-83 displays settings for the History system. At the very top, you can enable History for the simulators that support it. These are **Athena** and **Victory Process**. Then, there are two settings to select: **Length** and **Skip**. **Length** defines the maximum number of history files that are being kept. **Skip** defines the number of lines to skip between any two history points. A value of 0 for **Skip** means to save a history after every line of deck.

Below that, you can find the settings for the **File removal policy**. Changing these settings affects the way how DeckBuild cleans up when it exits. If set to **Always**, then all saved files are removed before DeckBuild exits. Setting to **Confirm** will open a confirmation dialog box (Figure 3-84). Setting to **Archive** will open a file selection dialog to name an archive file when DeckBuild exits. Setting to **Never** will keep the files unconditionally.

At the bottom, you will find settings that influence the status lines on the main editor window. The **Check size of files every** setting allows you to define an interval of how often the size of files generated by simulations is determined. **Warn if size of files exceeds** allows you to define a warning limit to detect cases of exhaustive use of disk space.

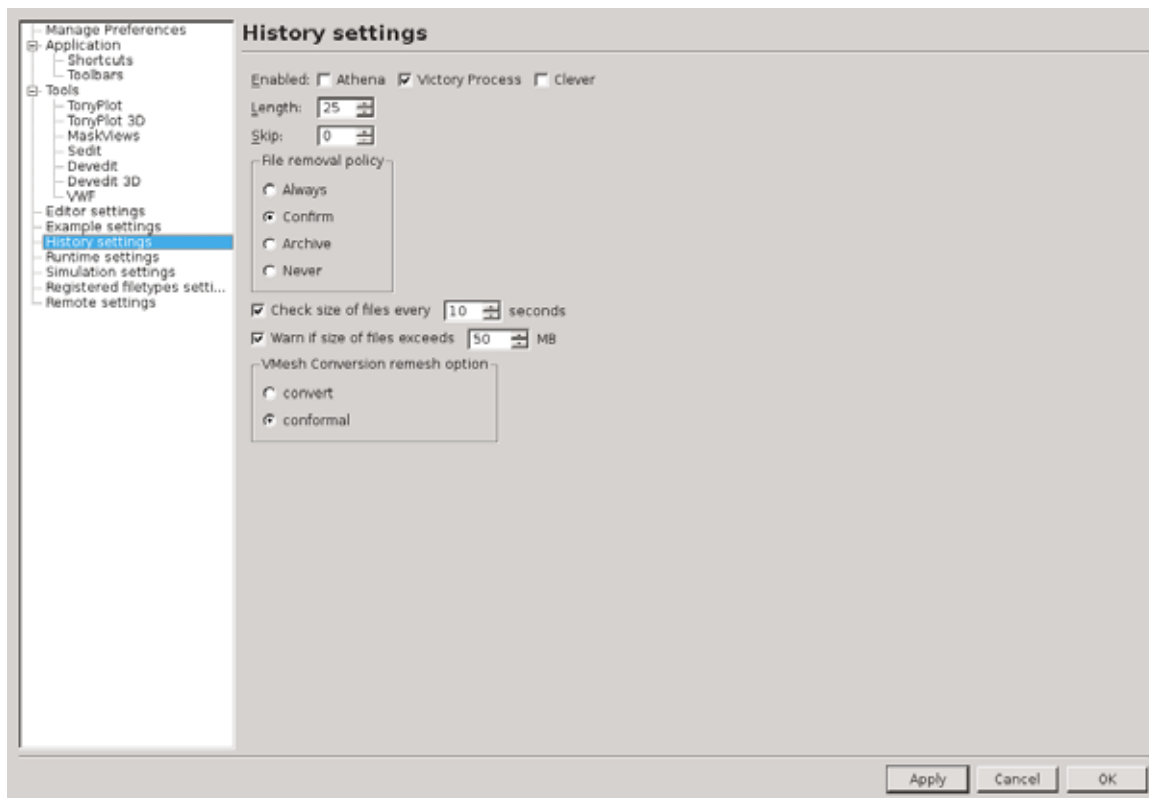


Figure 3-83 History Settings

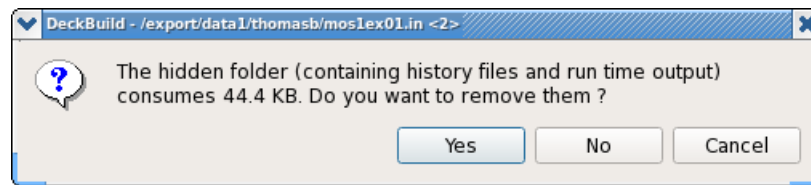


Figure 3-84 Dialog to confirm removal of files

3.28 Runtime Settings

Figure 3-85 shows the Preferences area that configures the runtime settings. The setting at the very top allows you to decide whether the standard error and standard output of a simulator will appear in a single pane or in separate panes in the main window. Standard error (**Error**) is normally used by simulators to indicate certain error conditions (e.g. a licensing problem). Standard output (**Output**) is used for the regular simulator output (e.g., to indicate the progress of the simulation).

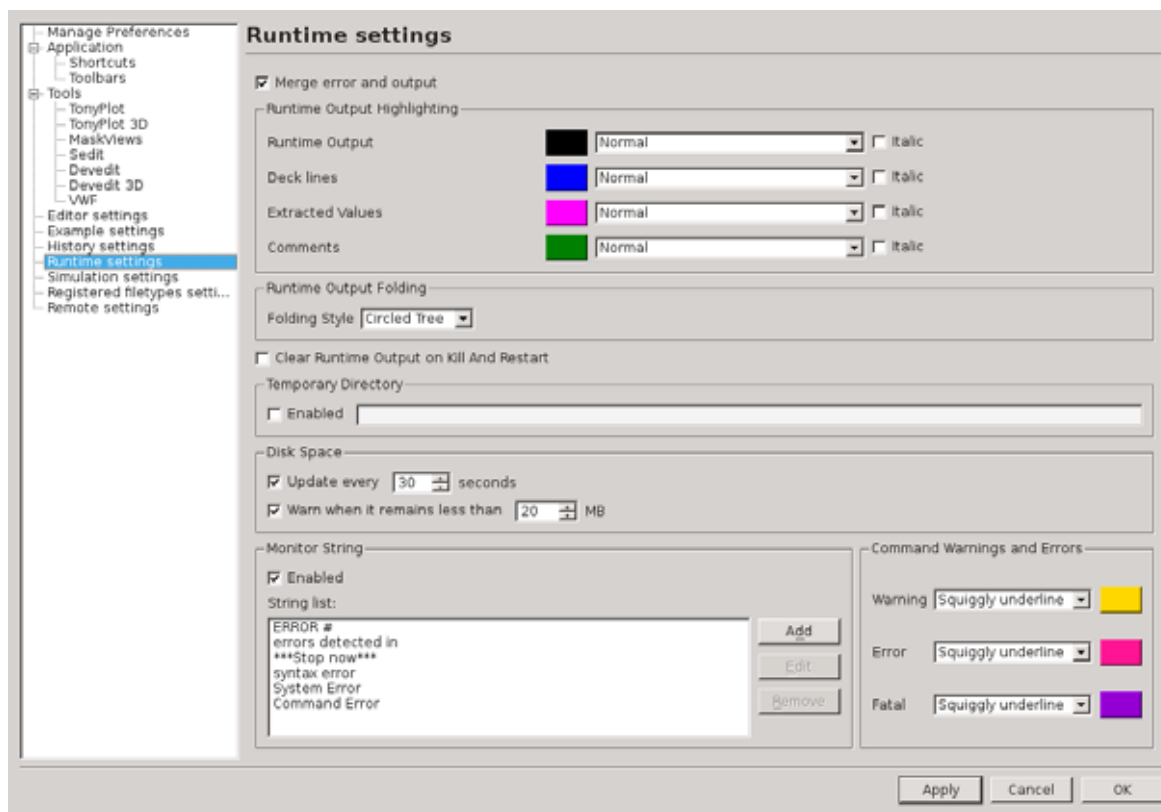


Figure 3-85 Runtime Settings

When "Clear Runtime Output on Kill and Restart" is ticked, Deckbuild will clear the runtime output and removed any text, which is stored from a previous simulation run.

The "Command Warnings and Errors" setting at the bottom right will enable syntax highlighting for simulators, which support the enhanced simulator interface. At this time this is the VictoryProcess 3D process simulator. The simulator supports three different warning/error stati ranging from 'Warning', over 'Error' to 'Fatal'. Note, that the meaning of a 'Warning' in this context is that the simulation does not stop. The only effect is the syntax highlighting indicated in the deck. The meaning of 'Error' is that the simulation will actually be stopped at the statement, which caused the error. The simulator will be kept running such that the user can correct the error and re-run the statement. Finally, a state of 'Fatal' means that the simulator either has crashed or will be terminated by deckbuild. The simulation cannot be continued.

Figure 3-86 shows the bottom left of the editor window in case standard output and standard error are shown in separated panes.

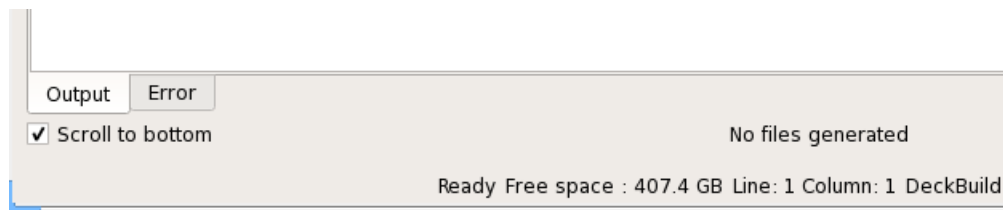


Figure 3-86 Standard Output Separate from Standard Error

The next group of settings is concerned what fonts and colors will be used to render the runtime output. Using different colors allows you to distinguish original deck lines from simulator response and extracted values. Figure 3-87 shows examples for all three available settings. The light grey rendered font indicates output from the simulator. The black font denotes deck lines (they are repeated in runtime output for clarity), whereas pink colored lines denote extracted results (gateox in this example).

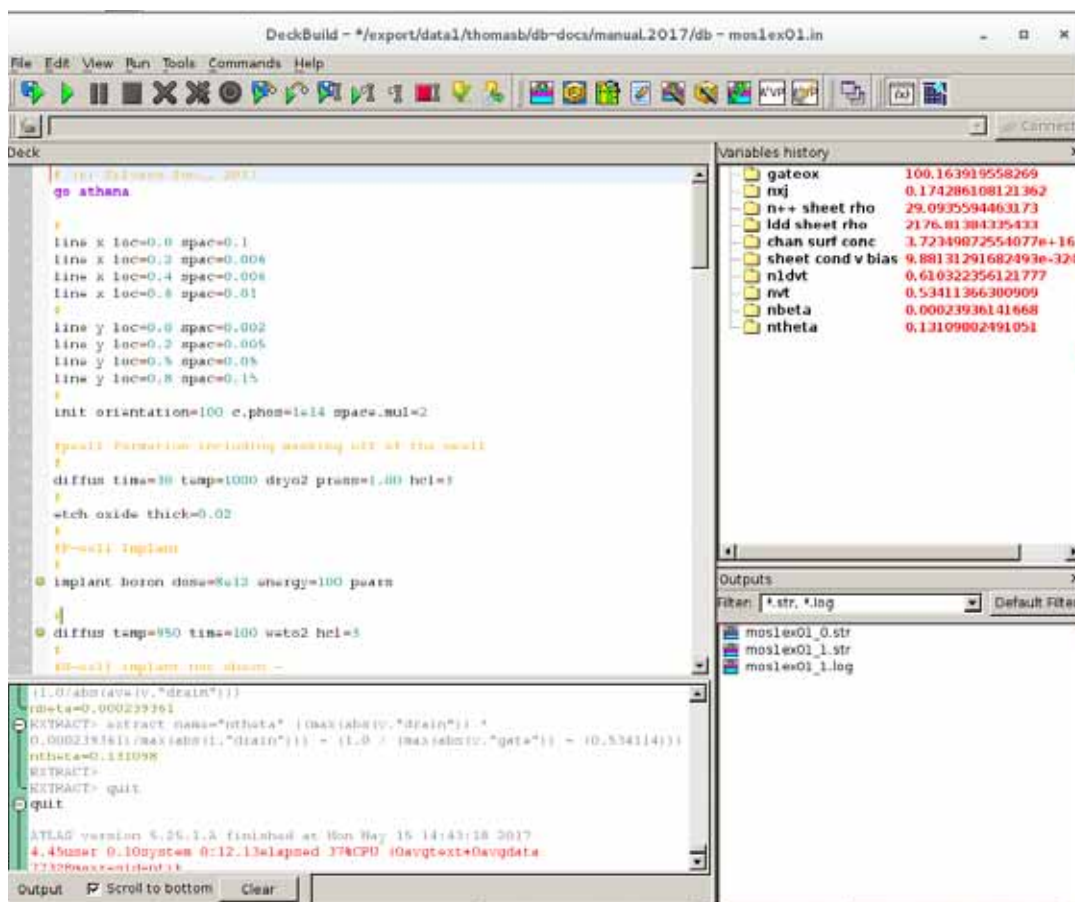


Figure 3-87 Formats Applied to Various Types of Runtime Output

The **Temporary Directory** setting allows you to choose another than the default temporary directory (usually /tmp on UNIX).

The **Disk Space** settings allow to define an interval how often the disk statistics are updated as well as a warning to get an indication in case your disk space drops below a certain mark.

Finally, the **Monitor Strings** allow you to define a list of strings to be watched by DeckBuild. As soon one of the defined strings appears anywhere in the runtime output, the simulation is immediately stopped and dialog will appear to indicate the problem. [Figure 3-88](#) shows that an illegal `diffus` command (`diffus error`) was entered. This triggers an error and a dialog will appear.

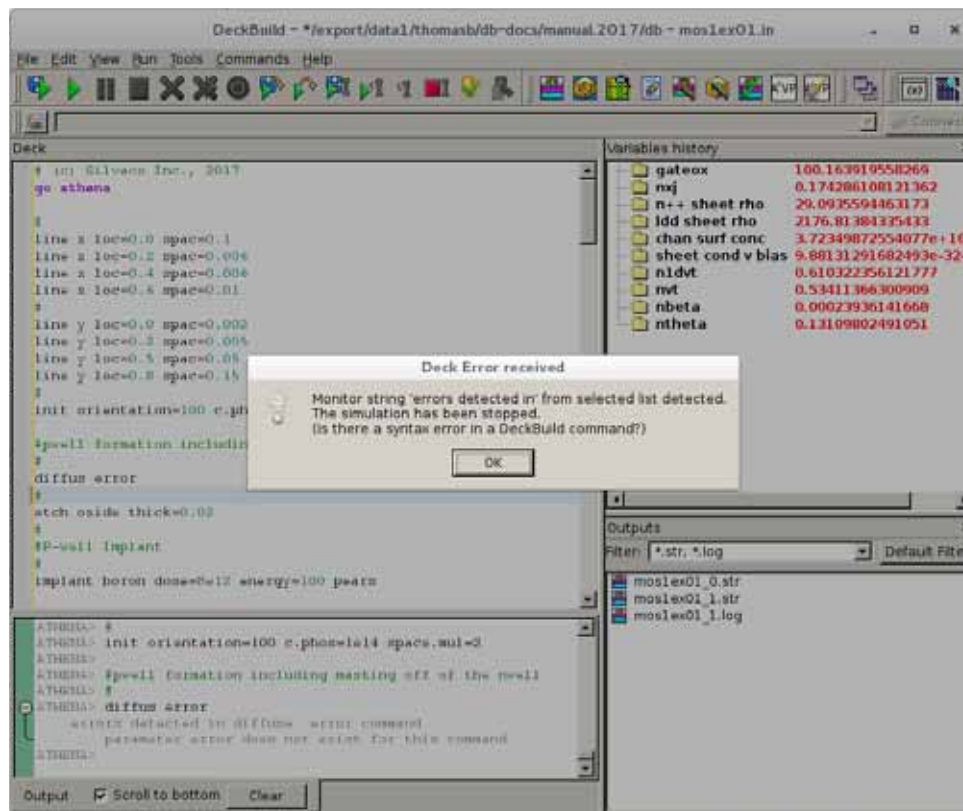


Figure 3-88 Detection of Monitor String

3.29 Simulation Settings

Figure 3-89 shows the preferences area for the Simulation settings.

The settings are split into two main areas. The first area has settings common to all simulators. This is the **nice** level that is being used to run a simulation. You can also disable the system commands (shell call out from within a deck) and the auto interface. The second area is dedicated to fine tuning where a simulator is started from and what version of the simulator you are using. When the **Use default simulator settings** box is checked, the whole area is greyed out and you cannot make any changes. However in Figure 3-89, the box has been unchecked. This allows you to make the following changes:

- Change the path of where a simulator is started from. If you click on the **Use default path** checkbox, then the path for the simulator is the same as where you started DeckBuild from. So if you installed your tools in, for instance, `/opt/silvaco`, then the simulators will be started exactly from there. If you happen to have several installations, for instance, to allow testing a particular release before installing it in the final location, then you can uncheck the **Use default path** box. This will enable the **Browse** button as shown in Figure 3-90. The selected directory needs to be a Silvaco installation. If you select a folder that does not contain the silvaco installation a dialog will appear (see Figure 3-91).
- Select a particular version. Independent of the selected install tree, you can choose among one of several versions of a simulator that are installed in your system.
- Using the **Override** simulator command allows you to run an arbitrary command instead of using a simulator. This option enables you to use a shell wrapper to start a simulator.

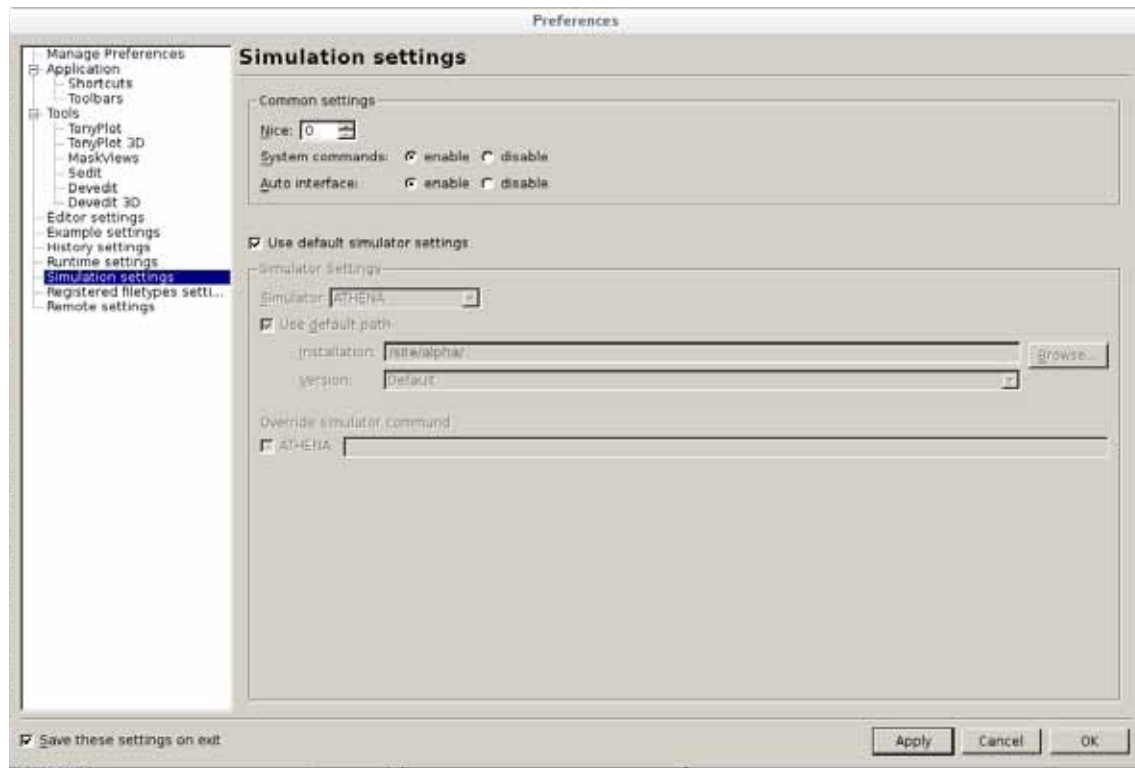


Figure 3-89 Simulation Settings

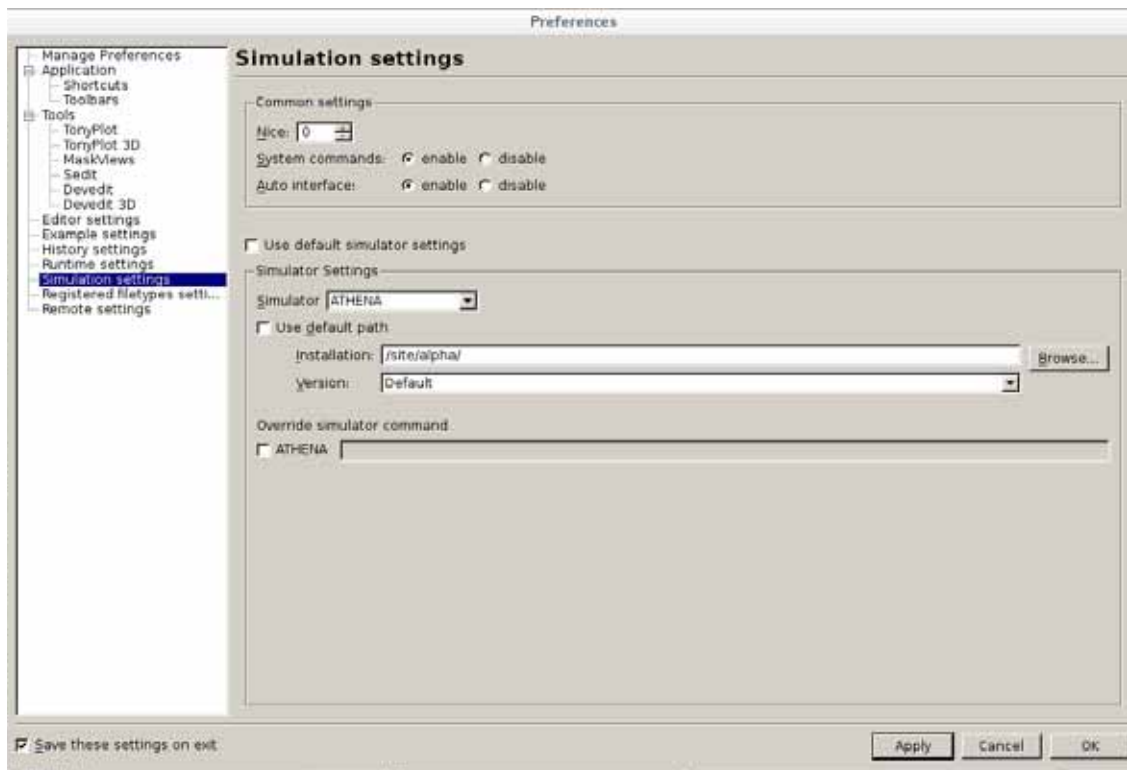


Figure 3-90 Selecting an Installation Path

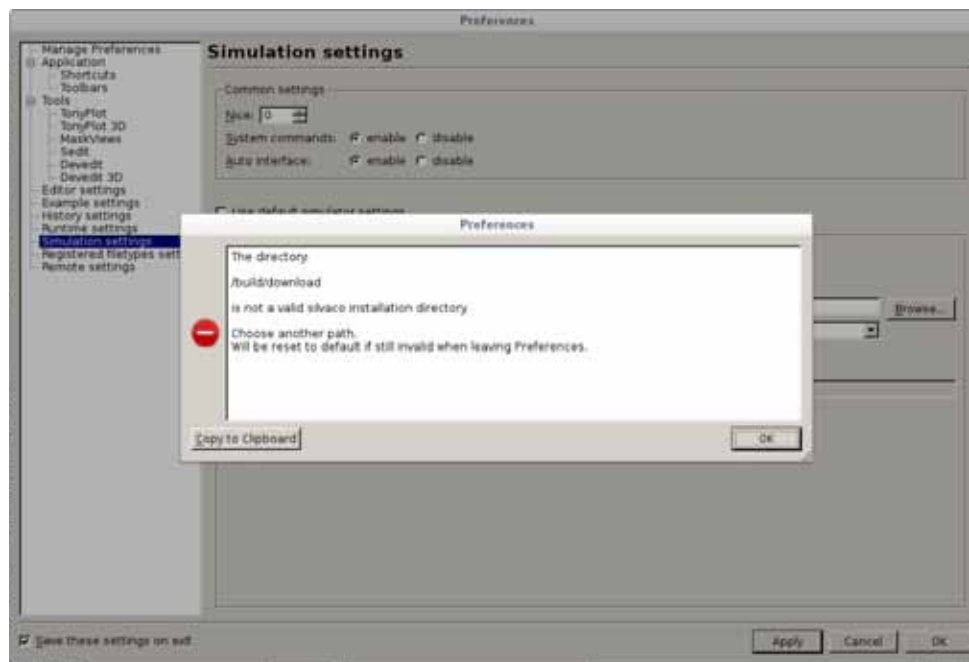


Figure 3-91 Illegal Installation Path Selected

3.30 Registered Filetypes

This area allows to associate viewer applications with file types based on the extension of a file.

3.31 Remote Settings

Please refer to [Section 3.3 Remote Mode](#) for full details.



Chapter 4 Statements

4.1 Overview

This section contains a complete description of every statement and parameter used by DeckBuild. The following information is provided for each statement:

- The statement name
- The syntax of the statement with a list of all the parameters of the statement and their type
- A description of each parameter
- An example of the correct usage of each statement

4.1.1 DeckBuild Commands

The following list identifies the commands that DeckBuild executes. Each of these commands is described in subsequent sections:

- ASSIGN
- AUTOELECTRODE
- DEFINE
- ELSE
- EXTRACT
- GO
- IF
- IF .END
- L .END
- L .MODIFY
- LOOP
- MASK
- MASKVIEWS
- SET
- SOURCE
- STMT
- SYSTEM
- TONYPLOT
- UNDEFINE

4.2 ASSIGN

Provides a much richer version of the functionality provided by the existing SET statement (see [Section 4.11 “SET”](#)).

Syntax

This is the syntax of the ASSIGN statement:

```
assign name = <variable> [print]

      (n.value = <expr_array> [delta=<expr> | ratio=<expr>] |
       l.value = <expr_array> |
       c.value = <qstring> [delta=<expr>] |
       <c_array>
      )

      [level = <expr>]
```

with the following subsidiary definitions :

```
<expr_array> -> <expr> |
                (<expr>, <expr_array>) |
                (<expr> <expr_array>)

<c_array>     -> c<integer>=<qstring> |
                c<integer>=<qstring> <c_array>
```

Description

The ASSIGN statement allows you to assign either a numerical (n), a logical (l) or a character (c) value to a variable. Numerical values may be arbitrary arithmetical expressions and may incorporate any of the standard functions mentioned in [Section 4.11 “SET”](#). All user-defined variables will be substituted before the expression is evaluated.

Arbitrarily, many variables may be assigned in the same deck.

Logical values may also be arbitrary numerical expressions. If any expression evaluates to a non-zero value, it is interpreted as true. Otherwise, it is interpreted as false. You can use the actual words "true" and "false". You can also assign arbitrary boolean expressions to logical values. The following operators are recognized:

```
logical AND      &

logical OR       |

logical NOT      ^
```

The usual relational operators are also recognised (>, <, >=, <=) with a single '=' character for the equals operator and the token ^= for the not-equals operator.

Note: Although unquoted strings are supported, you should **always** use quoted strings for character values for the sake of clarity.

You can assign a whole array of values to a variable. Arrays of numerical and logical arrays are written in the following manner:

```
(1, 2, 4, 8)
```

but arrays of character variables are written like this :

```
c2 = "Mary" c3 = "had" c5 = "a" c7 = "little" c11 = "lamb."
```

You can have many terms in a character array with their defining integers (the ones prefixed with 'c' for 'character') and not be sequential.

The array will be sorted in the increasing order of its defining integers.

Arrays are usually assigned to variables in loops. After each loop, the next value in the array will be assigned to the variable. If the end of the array comes before the end of the loop, the variable will revert to the first value in the array on the next pass.

You can also use the delta and ratio clauses to alter a variable on each pass through a loop. If you specify delta, that value will be added to the variable on each pass. If you specify ratio, the variable will be multiplied by that value on each pass.

If you specify an array of values, you cannot then specify either the delta or the ratio clauses.

You can specify a delta clause for a character value. This increment must be an integer and will be truncated if it isn't. This is an odd concept but is useful when, for example, you want to use a new output file on each iteration of a loop. A few examples will illustrate the idea. If the character value is a00 and delta is 4, then the first few values the variable takes will be a00, a04, a08, a12 and so on. Eventually, you will reach the values a92, a96, b00, b04, and so on. Incrementing lower-case 'z' by one produces lower-case 'a' but not upper-case 'A' and vice versa. You can also specify a negative delta with the obvious results.

An ASSIGN will persist until you encounter a second ASSIGN with the same variable name. If this happens, the old ASSIGN will be discarded and replaced by the new one. If an ASSIGN is outside of all loops, then the value of its variable never changes. If it's inside a loop, then its variable changes every time a new iteration of the loop begins.

If you specify the print keyword, the current value of the variable will print when initialized and will change each time thereafter.

You can use the level clause to have the value of the variable change when a particular member of a set of nested loops begins a new iteration. If the level you specify is positive, the loop is obtained by counting downwards from the zero level, the one outside of all loops. If the level is negative, the loop is obtained by counting upwards from the current level towards the outermost loop. So, level=-2 means change when the loop two above the present one starts a new iteration. level=2 means change when the next-to-the-outermost loop begins a new iteration.

As already mentioned, user-defined variables will be substituted before attempting expression evaluation. These variables are defined using the SET and ASSIGN statements. You can indicate the presence of a user-defined variable by prefixing it with '\$' or '@' or by surrounding it with braces like this:

```
 ${my_variable_1}, @{my_variable_2}.
```

Variables embedded within quoted strings will be correctly substituted. "Bare" variables will be recognized provided they are surrounded by both spaces and parentheses. This usage, however, is very confusing and highly inadvisable.

Examples

1. In this example, param1 will take the values 1, 2 and 3 on the three passes through the loop.

```
loop steps=3
  assign name=param1 print n.value = 1 delta = 1
1.end
```

2. This generates the sequence aa.20, aa.16, aa.12, aa.08, aa.04 and aa.00 for param2.

```
loop steps=6 print
  assign name=param2 c.value = "aa.20" delta = -4
1.end
```

3. Followed by, "Mary", "had", "a", "little" and "lamb".

```
loop steps=5 print
  assign name=param3 c10="lamb." c3="Mary" c8="little" c4="had"
  c7="a"
1.end
```

In the two preceding examples, the double quotation marks will **not** be included when param2 and param3 are substituted into later expressions.

4. param1 takes the values 42, 38, 17, 42, 38.

```
loop steps=5 print
  assign name=param1 n.value = (42, 38, 17)
1.end
```

5. param1 takes the values 42, 45.2, 48.4, 51.6, 54.8.

```
loop steps=5 print
  assign name=param1 n.value = 42 delta = 3.2
1.end
```

6. param1 takes the values 42, 134.4, 430.08, 1376.26, 4404.02.

```
loop steps=5 print
  assign name=param1 n.value = 42 ratio = 3.2
1.end
```

7. This is a simple example illustrating the use of boolean expressions.

```
assign name=condition l.value = ($x > 0.0 & $y < 3.0)
```

If x and y represent coordinates, the value of condition will be true or false accordingly as the coordinates are in a required area of the structure. The value of `$condition` could then be used as input to an `IF` statement.

8. It is worth emphasizing that `ASSIGN` can be used for the simplest of cases. See the following example:

```
assign name=e_charge n.value=1.6e-19
```

4.3 AUTOELECTRODE

Defines layout-based electrodes.

Syntax

```
autoelectrode
```

Description

The `autoelectrode` command causes DeckBuild to submit electrode definition statements to the current simulator. The electrode name and positioning information will be taken from the MaskViews layout data.

Note: DeckBuild only remembers the electrodes specified within each mask. Therefore, an `autoelectrode` statement must be used for every mask layer where electrodes are defined. This defines multiple electrodes for a single `autoelectrode` statement within the current mask.

See

“IC Layout Interface” section

4.4 DEFINE and UNDEFINE

DEFINE replaces all subsequent occurrences of an identifier with a specified string.

UNDEFINE cancels this action. All DEFINE statements may be canceled at once by calling SET CLEAR, see [Section 4.11 "SET"](#).

Syntax

```
define <identifier> <rest_of_line>
undefine <identifier>
```

Description

The identifier should either be a quoted string or a well-formed identifier. That is, one which begins with a letter or an underscore and continues with an arbitrary sequence of letters, digits, underscores and periods.

Every time this token is identified thereafter, it will be replaced by the whole of the rest of the DEFINE statement from the end of the token down to the end of the line. This <rest_of_line> component may consist of any characters whatsoever.

You don't have to flag the presence of the defined (DEFINE) token using a '\$' or '@' prefix or any of the other methods mentioned in [Section 4.2 "ASSIGN"](#).

Substitution of a defined (DEFINE) token will persist until you encounter an UNDEFINE statement referencing the same token.

Substitution of defined (DEFINE) tokens will occur before each line is executed, unless the line begins with a % character. This also holds for the DEFINE and UNDEFINE lines themselves and has an odd corollary, which you can see in the examples section.

Examples

1. Here is a straightforward example:

```
define mypath /home/john_smith/tmp/logs
.
.
.
log outf=mypath/file1.log
.
.
.
log outf=mypath/file2.log
```

This pathology will define black as white.

```
define color black
.
.
.
define color white
```

To get the behavior you probably had in mind, do this :

```
define color black
.
.
.
%define color white
```

2. Something similar happens with the UNDEFINE command. In the next example, "black" is substituted for "color" in the UNDEFINE command and a no-op results.

```
define color black
.
.
.
undefine color
```

3. For an UNDEFINE to take effect, always use the '%' prefix. For example:

```
define color black
.
.
.
%undefine color
```

4.5 EXTRACT

Extracts information from the current simulation.

Syntax

```
extract extract-parameters
```

Description

The `extract` statement is used to extract interesting information from the current simulation. See [Chapter 5: “Extract”](#) for a complete description.

4.6 GO

Interface between simulators

Syntax

```
go <simulator> [inflags=<> | outflags=<> | simflags=<> | cut-
line=<>|noauto]
```

Description

The GO statement tells DeckBuild to shut down the current simulator and start up the specified simulator when the statement is executed. It is used to auto-interface between simulators.

simulator can be `ssuprem3`, `athena`, `atlas`, `devedit`, `utmost`.

inflags specifies new load command flags for `autointerface`.

outflags specifies new save command flags for `autointerface`.

simflags specifies flags to be appended to default simulator argument.

cutline specifies a MaskViews cutline file to be loaded into DeckBuild.

noauto specifies that no `autointerface` occurs for this `go` statement.

Examples

If the current simulator is `SSuprem3`, then this statement causes DeckBuild to quit `SSuprem3` and start up `Athena`.

```
go athena
```

This will replace the default flags used in `Athena` auto interface command with `"master"` when loading and `"flip.y"` when saving.

```
go athena inflags=master outflags=flip.y
```

Note: One or more flags can be specified on the `go` line.

This statement will append `"-V 2.2.1.R"` to the default `DevEdit` argument to start version 2.2.1.R of the tool.

```
go devedit simflags="-V 2.2.1.R"
```

Note: Quotes are required where spaces used in flags or multiple flags used.

This loads the MaskViews cutline `default.sec` from the specified directory into DeckBuild.

```
go athena cutline="/usr/jdoe/default.sec"
```

This removes the currently loaded MaskViews cutline.

```
go athena cutline=none
```

Note: The cutline flag should never be used with VWF.

The cutline flag cannot be used within VWF because is no guarantee that the specified directory path for the cutline file will exist on any of the remote machines in a network that VWF jobs can be sent to.

If the current simulator is Athena, then the following statement causes DeckBuild to quit Athena and start up Atlas but no `autointerface` between the two simulators will occur.

```
go atlas noauto
```

See

“Auto Interface” section

4.7 IF, ELSE and IF.END

These three commands together provide the standard IF block functionality.

Syntax

```
if cond = (<boolean_expr>)  
  else [cond = (<boolean_expr>)]  
if.end
```

Description

The IF command starts the block. If its condition evaluates to true, then statements down to the next ELSE or IF.END line will be executed. If the condition evaluates to false, then there will be a search for an ELSE IF line whose condition evaluates to true. If you find such a line, the lines in its sub-block will be executed. At most, one sub-block in a given IF block will be executed.

The <boolean_expression> can be an arbitrary combination of boolean variables concatenated with AND, OR or NOT operators as described in [Section 4.2 “ASSIGN”](#).

You can nest IF blocks with each other and with LOOPS. As usual, an ELSE or an IF.END is associated with the most recent IF. There is no mechanism for using brackets or braces to enforce a particular nesting.

Example

```
if cond = (@MOSTYPE = "PMOS")  
  method gummel carriers = 1 holes  
  
else  
  method gummel carriers = 1 electrons  
  
if.end
```

4.8 LOOP, L.END and L.MODIFY

These three commands together provide the standard looping functionality.

Syntax

```
loop          steps = <expr> [print]

l.end        [break]

l.modify      [level = <expr>] [steps = <expr>] [next | break]
[print]
```

Description

Every `loop` statement must have a corresponding `l.end` statement. All the commands between these two statements are executed repeatedly for the number of times given in the `steps` clause of the `loop` command. If you specify the `print` keyword, the values of all user-defined variables that vary under the control of the loop will print every time they change. If you specify the `break` keyword in the `l.end` statement, the loop will exit on its first iteration regardless of the value of `steps`.

Example: Simple Loop In DeckBuild and Atlas

This example creates a simple resistor in Atlas and uses the loop functionality in DeckBuild to run two voltage solutions at 0.1V and 0.2V in Atlas.

```
go atlas

mesh

x.m l=0.0 s=0.01
x.m l=0.1 s=0.01

y.m l=0.0 s=0.01
y.m l=0.1 s=0.01

region num=1 silicon

electrode name=top top

electrode name=bottom bottom

doping num=1 conc=1e17 n.type uniform
```

```
solve init
solve previous

set a=0.1
loop steps = 2

solve v1=$a
set a=$a*2

l.end

quit
```

Loops can of course be nested with each other and with `IF` blocks. When an `l.end` statement is encountered, it is associated with the most recent loop statement.

The `l.modify` statement changes the behavior of the current loop or one within which it is nested. You specify the level of the loop you wish to modify using the `level` clause, which is described in [Section 4.2 “ASSIGN”](#). Without this clause, the current loop is assumed. You use the `steps` clause to change the number of times the loop will be executed. A value less than or equal to the current loop iteration count is acceptable and simply results in the loop exiting at the end of the current iteration.

The `break` keyword causes the loop to exit immediately.

The `next` keyword causes the loop to abandon the current iteration and to begin the next without executing any statements between the `l.modify` and the relevant `l.end` statements.

The `print` command switches on the printing of user-defined variables as described above.

Example

```
loop steps=3

    assign name=param1 print n.value = 1 delta = 1

loop steps=3
    assign name=param2 print n.value = 1 delta = 1
l.end

l.end
```

4.9 MASK

Defines the position of the process flow where photoresist or barrier material is added with the use of the MaskViews IC layout interface.

Syntax

```
mask name="maskname" [misalign=<misalignment>/
|bias=<bias>|delta_cd= <delta_cd>/
|shrink=<shrink>|reverse|optolith]
```

Description

Mask is used to interface to Silvaco's general purpose layout editor MaskViews. The `mask` statement defines the location where photoresist is deposited in the flow of processing events. The etched pattern is dependent on the MaskViews cutline file, which must be loaded into DeckBuild.

Name specifies the name of the layer that defines the photoresist patterning. This name must correspond to a mask level name contained in the MaskViews cutline file loaded into DeckBuild.

Bias and delta_cd increase or decrease the width of the deposited mask. For positive masks, a positive `delta.bias` decreases the etched hole(s) in the mask.

Misalignment shifts the entire specified mask left and right. Negative misalignment values shift the mask left, positive values right.

Shrink reduces the size of the specified layer by the ratio specified.

Reverse specifies that the mask polarity should be reversed or that negative type photoresist should be modeled.

Optolith specifies that the loaded MaskViews cutline is from an Optolith layout. Therefore, OPTOLITH syntax (layout commands) is used to define the photoresist pattern.

Examples

The `delta` value can be used to vary the Critical Dimension (CD) of the specified layer. The value operates on an edge-by-edge basis. For example, for an IC layout with a 1.0, micron wide "poly" the statement:

```
mask name="poly" delta=-0.1
```

creates a drawn poly length of 0.8 microns, meaning that 0.1 have been removed from each poly edge.

The `bias` command option performs the same operation as the `delta` command. This can be used globally to edit the bias of each layer. The `bias` command can be used with `delta`, such that the real value for CD reduction is the sum of the `delta` and `bias` values, per edge. For example, if an IC layout with 1.2 micron CD's is streamed-in from GDS2, and the final etch, then the final etch profile is known to be 0.9 microns due to a combination of biasing, photo-exposure, and over etch, then the offset is required to be constant. This is where the `bias` command can be used.

```
mask name="poly" bias=-0.15
```

In other words, 1.2 microns-0.9 microns=-0.3 microns =2(-0.15) microns, or -0.15 microns per edge.

Further experimentation might be required in addition to the fixed bias. This is where the `delta` command can be used. In this example:

```
mask name="poly" bias=-0.15 bias=-0.15 delta=-0.025
```

This simulates a true experiment in terms of CD variation.

The `misalign` command is used to offset a layer with respect to other layers. For example

```
mask name="poly" bias=-0.15 misalign=-0.1
```

causes the poly layer to be offset to the left by 0.1 microns.

The `shrink` command is used to reduce the size of all edges in the specified layer. For example, the statement below will reduce the layer edges by 50 percent.

```
mask name="poly" shrink=0.5
```

Misalignment and CD Experimentation

It is often necessary to experiment with either misalignment or the CDs of a layer. The MaskViews-DeckBuild interface supports this level of experimentation. DeckBuild can be used to experiment with the outline generated by MaskViews. Each `mask` statement can be used to alter the outline. The underlying mesh used by Athena is not changed with mask experimentation commands. VWF can be used to split on these values to generate RSM's relating to mask experimentation.

4.10 MASKVIEWS

Plots a layout file

Syntax

```
maskviews <layout file>
```

Description

This statement starts the MaskViews layout editor and load the supplied layout file. If no layout file is specified, MaskViews is invoked with no data.

Examples

This statement plots a layout file (which should be in the current directory).

```
maskviews layout.lay
```

See

MaskViews User's Manual

4.11 SET

Sets the value of a user-defined variable or clear all existing variables.

Syntax

```

set <variable> = <value> | <expr> | <built_in_func> [nominal]
set clear
<built_in_func> = max    (<expr>, <expr>) |
                  min    (<expr>, <expr>) |
                  ave    (<expr>, <expr>) |
                  sin    (<expr>)          |
                  cos    (<expr>)          |
                  tan    (<expr>)          |
                  asin   (<expr>)          |
                  acos   (<expr>)          |
                  atan   (<expr>)          |
                  atan2  (<expr>, <expr>) |
                  sinh   (<expr>)          |
                  cosh   (<expr>)          |
                  tanh   (<expr>)          |
                  exp    (<expr>)          |
                  log    (<expr>)          |
                  log10  (<expr>)          |
                  pow    (<expr>, <expr>) |
                  sqrt   (<expr>)          |
                  ceil   (<expr>)          |
                  floor  (<expr>)          |
                  abs    (<expr>)          |
                  ldexp  (<expr>, <expr>) |
                  fmod   (<expr>, <expr>)

```

Description

The `set` command is used to set the value of a user-defined variable. The value can later be substituted using `$`-substitution, which replaces the variable name with its value when the variable is preceded by a dollar sign `'$'` or by the at sign `'@'`.

variable is a user-defined variable name. It may contain spaces or other non-alphabetical characters if it is delimited by double quotation marks..

<expr> is an algebraic expression consisting of numeric constants, `$`-substituted variables, algebraic operators (+, -, *, /, ^), and or the built-in functions shown.

set commands can be used in conjunction with extracted values. If a `$-variable` is to be substituted and if an existing DeckBuild variable cannot be found, it is assumed to be a user-defined environment variable.

Synonyms for SET

We have introduced synonyms for the SET syntax to provide improved compatibility with other products. Both of the following syntaxes are valid.

```
Assign|assign NAME=<variable> N.VALUE= <value> | <expr> |
<built_in_func>
```

```
define <variable> <value> | <expr> | <built_in_func>
```

These will assign the chosen value or expression to the variable in the same way as the existing SET syntax. To reiterate, you can substitute the value later using `$-substitution`, which replaces the variable name with its value when the variable is preceded either by a dollar sign '\$', or by the at sign '@'.

Examples

These statements show how to set variables and how to substitute with each other and in simulator syntax.

```
set time=30
set temp=1000
set press=1.0
set env="nitro"
set pi=3.1415
set "pi*2" = 2*$pi
```

```
diffuse time=$time temp=$temp press=$press $env hcl="$pi*2"
```

The following statements extract the thickness of the top layer of oxide in a structure and etch back that thickness plus 0.05 micron.

```
extract name="oxide thickness" thickness oxide
set etch_thickness = ("oxide thickness"*10000) + 0.05

etch oxide dry thickness=$etch_thickness
```

Note: The thickness is measured in angstroms, so it is converted to microns first.

Variable names that contain spaces (generated by `extract` statements) must be quoted for `$-substitution`, and the ``$'` must precede the quoted string as shown.

For variable names with no spaces, quotation marks are optional.

The following statement will remove all existing variables, including those made by `define` statements, see [Section 4.4 "DEFINE and UNDEFINE"](#).

```
set clear
```

The statements below show the use of the nominal flag.

```
extract name="oxide thickness" "oxide thickness" thickness_bad_syn-  
tax oxide  
set "oxide thickness" = 0.5 nominal  
etch oxide dry thickness=$oxide thickness
```

For this example, if the `extract` statement was successful, the value of "oxide thickness" would be set. Therefore, the `nominal set` statement would be ignored. But the `extract` syntax is incorrect, so the `extract` statement never creates the result variable and "oxide thickness" is set by the `nominal set` statement to 0.5.

4.12 SOURCE

Enables simulation commands to be executed from an external file

Syntax

```
SOURCE file
```

Description

The `SOURCE` statement enables simulation commands to be executed from an external file. The named file is read and placed in DeckBuild's input buffer and is executed as if it were part of the input deck.

`file` is the name of a file that contains any valid simulator syntax or DeckBuild statements, such as `extract` and `set`. The sourced file may source other files. If the file name does not begin with `'/'`, then it is assumed to be in the current directory.

Examples

The file to be sourced may contain part of an input deck including commands from any simulator. The following input deck fragment will perform a diffusion, access the file `include_file` for further commands, then revert back to the deposition step.

```
etch oxide all
#
# Source an external file
# Return to the input deck
implant bf2 dose=1.0e12 energy=35 pearson
include_file contains these statements:
#gate oxide grown here:-
diffus time=10 temp=900 dryo2 press=1.00 hcl%=3
```

The runtime output from this fragment will appear as:

```
ATHENA> etch oxide all
ATHENA> #
ATHENA> # Source an external file
ATHENA> source include_file
ATHENA> #gate oxide grown here:-
ATHENA> diffus time=10 temp=900 dryo2 press=1.00 hcl%=3
Solving time(sec.)      0 +      0.01      100%, np 106
Solving time(sec.)      0.01 +   0.173987   1739.87%, np 106
Solving time(sec.)    0.183987 +   0.187665   107.861%, np 106
*
Solving time(sec.)    0.371653 +   0.628347   334.823%, np 106
Solving time(sec.)      1 +      0.1      15.9148%, np 106
Solving time(sec.)      1.1 +   3.1396    3139.6%, np 106
```

```
Solving time(sec.)    4.2396 +    19.1813    610.947%, np 106
Solving time(sec.)    23.4209 +    93.4041    486.955%, np 106
Solving time(sec.)    116.825 +     150    160.593%, np 104
Solving time(sec.)    266.825 +     150    100%, np 104
Solving time(sec.)    416.825 +     150    100%, np 104
Solving time(sec.)    566.825 +    33.1751    22.1167%, np 104
ATHENA ># Return to the input deck
ATHENA> implant bf2 dose=1.0e12 energy=35 pearson
```

4.13 STMT

Enables you to define variables that change under the control of loops.

Syntax

```
stmt <parameters>
```

Where

```
<parameters> -> <parameter> | <parameter> <parameters>
```

```
<parameter> -> <variable> = <initial> [ : [ + | * ] <change> [ :  
<level> ] ]
```

That is, a `stmt` command must carry at least one parameter and may carry many (independent) parameters.

Description

This is effectively a shorthand for part of the `ASSIGN` statement.

The `<initial>`, `<change>` and `<level>` terms are all numerical expressions.

The value of the variable is re-evaluated **every** time the `STMT` command is encountered. If no arithmetical operator is specified or if the '+' sign appears explicitly, then addition is understood and the variable is re-evaluated as

$$\langle \text{initial} \rangle + \langle \text{change} \rangle * (\text{count} - 1)$$

where `count` is the current iteration count of the loop with level `<level>`.

If the multiplication operator ('*') appears, the variable is re-evaluated as

$$\langle \text{initial} \rangle * \text{pow}(\langle \text{change} \rangle, (\text{count} - 1))$$

where `pow` is the usual exponentiation function.

The `<change>` term defaults to 0 in the addition case and 1 in the multiplication case. The `<level>` term defaults to the current loop level. This means that if you only specify `<initial>`, the variable will be a constant.

Examples

1. In this example `param1` will take the values 1, 2, 3, 4 and 5.

```
loop steps=5 print
  stmt param1=1:1
1.end
```

2. In this example `param1` will take the values 1, 2, 4, 8 and 16.

```
loop steps=5 print
  stmt param1=1:*2
1.end
```

4.14 SYSTEM

Allows DeckBuild to execute UNIX system commands within a simulation deck.

Syntax

```
SYSTEM <UNIX command>
```

Description

The `SYSTEM` command allows you to execute shell scripts or perform other UNIX tasks directly from the simulation deck. The command is blocking, meaning that the simulation does not continue until the `SYSTEM` command has finished execution.

To use this feature, enable the `SYSTEM` commands in the Main Control Options Popup (see [Figure 3-18](#)). To enable system commands for VWF Automation Tools, set the environment variable `DB_SYSTEM_OPTION` to any value.

Examples

```
system rm history*.str
```

Note: Redirection of the system command output (i.e., `system ls * .in > file.out`) cannot be achieved as the output is already redirected by DeckBuild.

4.15 TONYPLOT

Plots a file

Syntax

```
tonyplot -args
```

Description

This statement causes DeckBuild to save a temporary file from the current simulator and start up TonyPlot with that file loaded. The temporary file is removed when TonyPlot exits.

`-args`, if specified, are passed directly to TonyPlot (as if invoked from the command line). If any of `-st`, `-da`, or `-over` and a file name is specified, DeckBuild uses the named file instead of saving and plotting the current structure.

DeckBuild also detects if the structure to be plotted is 3D and use TonyPlot3D if required.

Examples

This statement saves the current file and starts TonyPlot.

```
tonyplot
```

This statement plots a file (which should be in the current directory).

```
tonyplot -st well.str
```

See

[Section 3.4 “DeckBuild Controls”](#)



Chapter 5 Extract

5.1 Overview

DeckBuild has a built-in extraction language that allows measurement of physical and electrical properties in a simulated device. The result of all extract expressions is either a single value (such as X_j for process or V_t for device), or a two-dimensional curve (such as concentration versus depth for process or gate voltage versus drain current for device).

Extract forms a “function calculator” that allows you to combine and manipulate values or entire curves quickly and easily. You can create your own, customized expressions, or choose from a number of standard routines provided for the process and device simulators. You can take one of the standard expressions and modify it as appropriate to suit your needs. Extract also has variable substitution capability so that you can use the results of previous `extract` commands.

Extract has two built-in 1D device simulators, QUICKMOS and QUICKBIP, for specialized cases of MOS and bipolar electrical measurement. Both QUICKMOS and QUICKBIP run directly from the results of process simulation for fast, easy and accurate device simulation.

5.2 Process Extraction

DeckBuild's process extraction window is shown below (Figure 5-1).

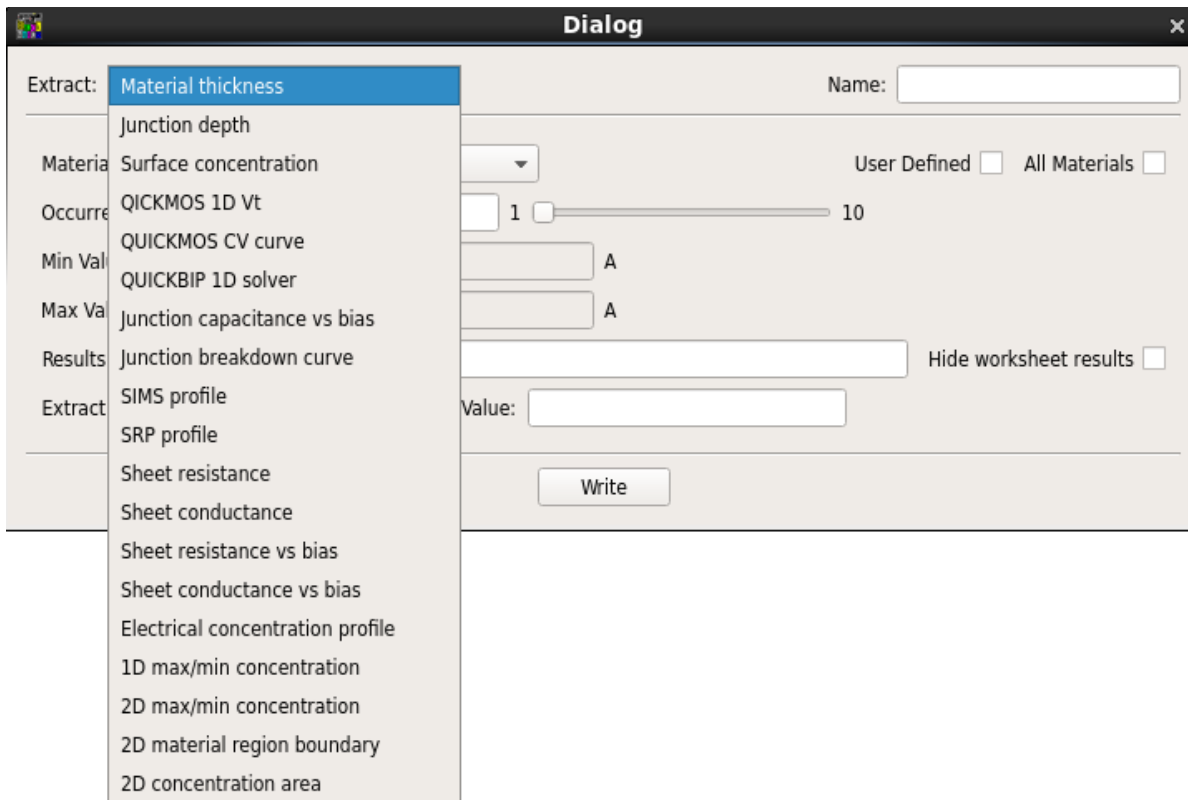


Figure 5-1 Process Extraction Dialog

You may use this window to look at the following:

- **Material thickness** measures the thickness of the nth occurrence of any material or all materials in the structure.
- **Junction depth** measures the depth of any junction occurrence in the nth occurrence of any material.
- **Surface concentration** measures the surface concentration of any dopant, or net dopant, in the nth occurrence of any material.
- **QUICKMOS 1D Vt** calculates the one-dimensional threshold voltage of a MOS cross section using the built-in QUICKMOS 1D device simulator. The gate voltage range defaults between 0 to 5 Volts but can be specified as required. The substrate can also be fixed at any bias. Qss and device temperature values may also be specified.
- **QUICKMOS CV curve** creates a CV curve of a MOS cross section using QUICKMOS. This shows capacitance as a function of either gate voltage or substrate voltage with the other terminal held at any fixed bias. Qss and device temperature values may also be specified.
- **QUICKBIP 1D solver** measures any of 22 BJT Gummel-Poon parameters, plus any forward or reverse IV curve. See the [Section 5.9 “QUICKBIP Bipolar Extract”](#) for more information and examples.

- **Junction capacitance versus bias** calculates the junction capacitance of a specified p-n junction within any region as a function of applied bias to that region. Qss and device temperature values can also be specified.
- **Junction breakdown curve** calculates the electron or hole ionization integral of any region as a function of applied bias to that region. This calculation uses the Selberherr impact ionization model. (see the Impact command section and Impact Ionization physics sections within the Atlas manual). You can modify the Selberherr model default values and specify Qss and device temperature values.
- **SIMS profile** calculates the concentration profile of a dopant in a material layer.
- **SRP profile** calculates the SRP (Spreading Resistance Profile) in a silicon layer.
- **Sheet resistance and sheet conductance** calculates the sheet resistance or conductance of any p-n region in any layer in an arbitrary structure. You can specify the bias of any region in any layer, the Qss of any material interface and the device temperature. A flag for carrier freezeout calculations can also be set (see the “Incomplete Ionization Of Impurities” physics section within the Atlas manual).
- **Sheet resistance and sheet conductance versus bias** calculates the sheet resistance or conductance of one or more regions as a function of applied bias to any region. Qss and device temperature values can also be specified.
- **Electrical concentration profiles** measures electrical distributions versus depth. You can also specify the bias of any region in any layer and the Qss of any material interface. The device temperature can also be set to the required value. The following distributions are calculated:
 - electrons
 - holes
 - electron quasi-fermi level
 - hole quasi-fermi level
 - intrinsic concentration
 - potential
 - electron mobility
 - hole mobility
 - electric field
 - conductivity
- **1D maximum/minimum concentration** measures the peak or minimum concentration of any dopant or net dopant, for a specified 1D outline, in the nth occurrence of any material or all materials, and also within any junction-defined.
- **2D maximum/minimum concentration** measures the peak or minimum concentration of any dopant or net dopant, for the whole 2D structure or within a specified area, in any material or all materials, and also within any junction-defined. The actual xy coordinates of the maximum or minimum concentration can also be retrieved.
- **2D material region boundary** returns the maximum or minimum boundary of the selected material region for either X or Y axis. Therefore, the outer boundaries of any material region can be extracted.
- **2D concentration area** integrates specified concentration of any dopant or net dopant for whole 2d structure or within a specified location.

- **2D maximum concentration file (CCD)** creates a **Data Format** file with the XY coordinates and the actual values of the maximum concentrations stepping across the structure. This file can be loaded into TonyPlot when in **-ccd** mode to show a line of maximum concentration across a device.
- **ED tree** creates one branch of a **Smile** plot or **ED** tree from multiple **Defocus** distance against **Critical Dimension (CD)** plots created for a sweep of **Dose** values by Optolith. These plots are all written in a single Data format file.
- **Elapsed time** extracts time stamps from a specified start time at any point in a simulation. You can reset the start time as required.

Note: This extraction is not CPU time.

The built-in 1D Poisson device simulator is used to calculate sheet resistance and conductance and the electrical concentration profiles.

With the exception of 2D extractions, all the process extraction routines are available from both 1D and 2D process simulators. In the case of the 2D simulators, a cross section x or y value or region name (used in conjunction with MaskViews) determines the 1D section to use.

Note: An error will be returned for attempted extractions on 3D structure files.

5.2.1 Entering a Process Extraction Statement

To place an `extract` statement in your process deck, select **Commands**→**Athena Extract...**. The **Extraction** popup appears. The popup for extracting Athena is shown in [Figure 5-2](#).




Figure 5-2 Thickness Extraction Dialog

Choose the extract routine you want by activating a choice on the **Extract** setting. The popup changes size and displays different items, depending on which routine you choose. Then, enter or choose the desired information for each item on the popup. An extract name is always required. Optionally, enter the minimum or maximum desired cutoff values by checking **Min value** or **Max value** and entering a value in the corresponding text field. By default, all extract results are written to a file named `results.final`. But using the **Results datafile** field allows you to specify the results file for each individual `extract` statement. Material and impurity names are selected using a **Pulldown menu** ([Figure 5-3](#)).

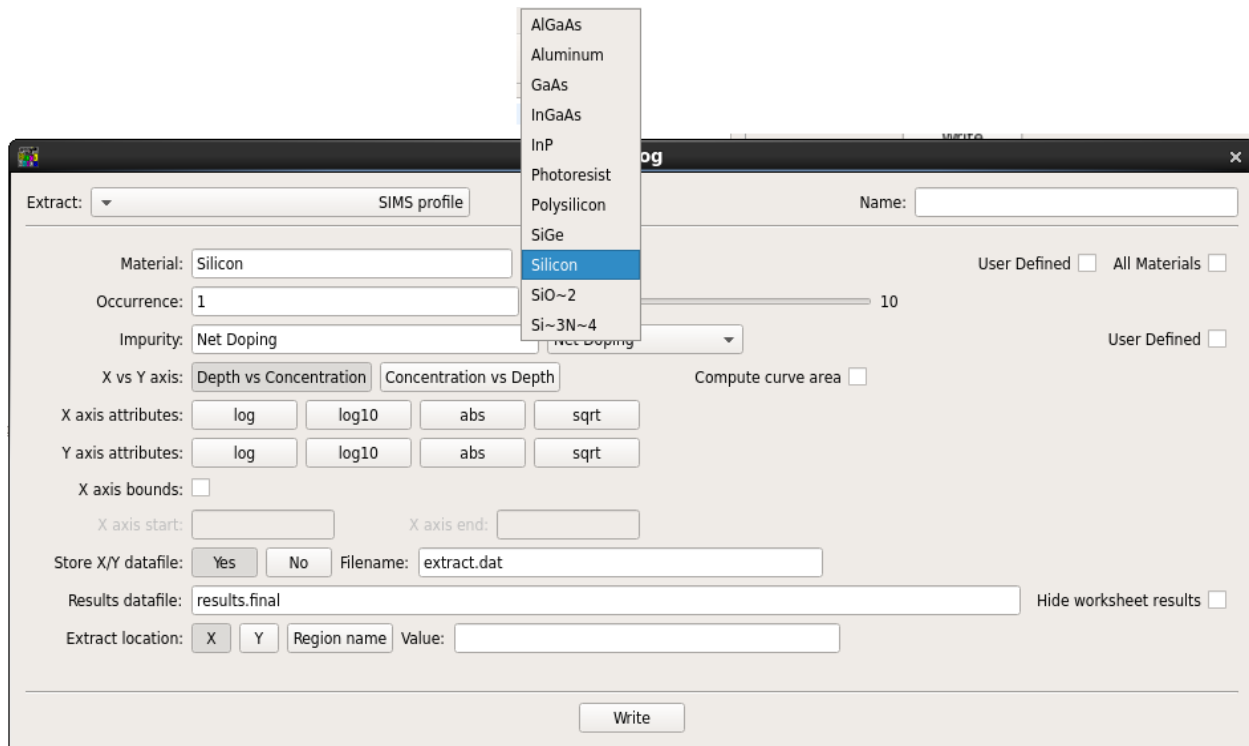


Figure 5-3 SIMS Profile Extraction Dialog Showing Material Pulldown

If the required option is not present in the default setting, select the **User Defined** checkbox to allow entering materials/impurities in the corresponding text box.

Finally, place the text caret at the desired point in the deck and click on the **WRITE** button. The extract syntax is written to the deck.

5.2.2 Extracting a Curve

Some of the process extraction statements create a two-dimensional curve as a result, rather than a single value. For instance, `extract` constructs a data set of concentration versus depth for the SIMS, SRP, and electrical quantities distributions. You can use the resulting 2D curve for measurement and testing and as a target on the Optimizer worksheet so that you can optimize against 2D curves.

Extract provides several additional options to 2D curve support: axis layout, axis attributes, optional computation of area under the curve, and optional outfile. These options are the same regardless what type of curve (for instance, QUICKMOS CV and SIMS profile) you are extracting.

The Athena Extract popup showing the SIMS Profile is shown in [Figure 5-4](#).

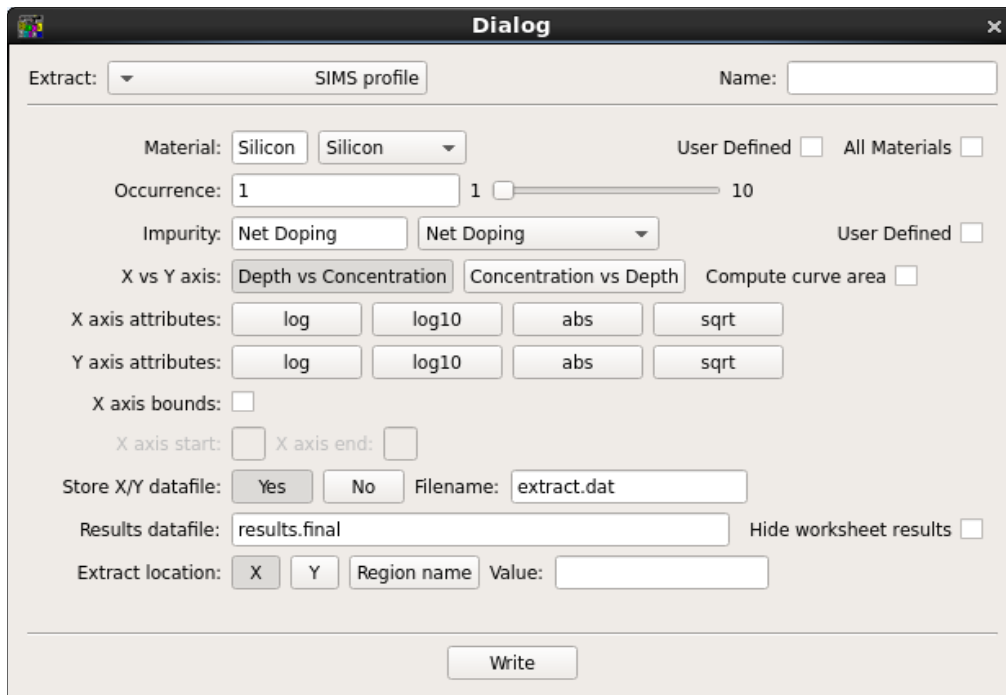


Figure 5-4 SIMS Profile Extract Dialog

The following options are available:

- **X vs Y axis** determines the x and y axes of the resulting profile curve. The default (which should always be used unless you plan to customize the resulting extract expression) is that the x axis is depth into the material, and the y axis is the concentration.
- **X axis attributes** and **Y axis attributes** allows you to modify the data values on each axis independently. To compute net concentration versus depth, you can select **abs** on the y axis (concentration), and select nothing on the x axis (depth). **abs** is always evaluated before taking the log or square root of the data.
- **Curve X axis bounds** specifies whether to create the curve for the whole X axis or for only a required section. If selected, **X axis value** fields become active, enter values in the same units as the resulting curve. This is useful for extracting local maxima and minima.

- **Store X/Y datafile** stores an output file in TonyPlot data format if set to **Yes**. You can plot the data file in TonyPlot using the `-da` option. You can also read the data file directly into the Optimizer worksheet as a target if desired.
- **Compute curve area** computes the area under the curve. When checked, it causes several other items to become active.
- **Area X axis bounds** tells Extract whether to integrate the area under the curve along its entire length or just for a bounded portion of the X axis. If you select **Bounded**, then **X axis start** and **X axis stop** become active. Enter **start** and **stop** values in the same units as the resulting curve.

To construct the 2D curve, set each item on the popup in turn and click on **WRITE**.

Depth is always computed as distance from the top of the selected material layer and occurrence. Depth starts from 0 and increases through the material.

5.3 Customized Extract Statements

In addition to the simple curve primitives shown on the popup, you can edit the input deck directly to make customized curves. Examples include extracting maxima and minima on the curve, combining axes using a function definition, looking at slopes of tangent lines, intercepts of sloped lines. The Extract syntax is described below, followed by examples of process extraction. See the examples listed under Section 5.4 “Device Extraction” for more information.

5.3.1 Extract Syntax

Text inside matching pairs of `/*` and `*/` delimiters are comments. These are used to clarify the meaning of the syntax and also as definitions for the most primitive types, such as `<QSTRING>`.

The backslash character (`\`) at the end of a line indicates a continuation line.

Many of the optional parameters (the ones enclosed in square brackets) have default values. Some of these defaults are given immediately after they appear. Others appear in more than one place and so are collected at the end.

Description

`<EXTRACT_STATEMENT>` :

`<EXTRACT_SINGLE_LINE_GENERAL>`

`<EXTRACT_MULTIPLE_LINE_GENERAL>`

`<EXTRACT_2D_MAX_MIN_CONC>`

`<EXTRACT_TIME>`

`<EXTRACT_SIMPLE>`

`<EXTRACT_SINGLE_LINE_GENERAL>` :

`[extract init infile=QSTRING]`

```
/* In default of the above line, a temporary structure file represent-
ing the current state of the device will be          constructed. */
extract [name=<QSTRING>] <EXTRACT_SINGLE_LINE_PARTICULAR> \
[datafile=<QSTRING>] [hide]
```

```

<EXTRACT_MULTIPLE_LINE_GENERAL> :

    [extract init infile=<QSTRING>]

    /* In default of the above line, a temporary structure file
    representing the current state of the device will be constructed. */

    extract start <EXTRACT_MULTIPLE_LINE_SETUP_N>

    [extract cont <EXTRACT_MULTIPLE_LINE_SETUP_N> ...]

    /* zero or more instances of the extract cont line may appear. */

    extract done [name=<QSTRING>] <EXTRACT_MULTIPLE_LINE_DONE_N> \
                [datafile=<QSTRING>] [hide]

    /* There are five pairs of definitions for<EXTRACT_MULTIPLE_LINE_SETUP_N>
    and <EXTRACT_MULTIPLE_LINE_DONE_N>, with N replaced by 1, 2, 3, 4 or 5.
    Elements from different pairs (ones with different values of N) must NOT
    appear in the same statement. */

<EXTRACT_2D_MAX_MIN_CONC> :

    [extract init infile=<QSTRING>]

    /* In default of the above line, a temporary structure file representing
    the current state of the device will be constructed. */

    extract [name=<QSTRING>] 2d.max.conc | 2d.min.conc [interpolate] \
    [<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \
    [min.v=<EXPR>] [max.v=<EXPR>] \
    [x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR> |
    y.max=<EXPR> y.min=<EXPR> region=<QSTRING>] \
    [datafile=<QSTRING>] [hide]

    [extract [x_pos_name=<QSTRING>] x.pos

```

```
extract [y_pos_name=<QSTRING>] y.pos]
```

```
/* x_pos_name and y_pos_name will default to the name in the|
main extract statement, with "X position" and "Y position
appended respectively. */
```

```
<EXTRACT_TIME> :
```

```
extract [name=<QSTRING>] clock.time [start_time=<EXPR>] \
[datafile=<QSTRING>]
```

```
<EXTRACT_SIMPLE> :
```

```
extract [name=<QSTRING>] <EXPR> [datafile=<QSTRING>]
```

```
<EXTRACT_SINGLE_LINE_PARTICULAR> :
```

```
<CURVE_FUNC> (<CURVE_SINGLE_LINE>)
[outfile=<QSTRING> [sigfigs=<EXPR>]]
```

```
<EXTRACT_MULTIPLE_LINE_DONE_5>
```

```
thickness [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>]
```

```
xj [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \
[x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] [junc.occno=<EXPR>]
```

```
surf.conc [min.v=<EXPR>] [max.v=<EXPR>] [<MATERIAL>] [mat.occno=<EXPR>] \  
          [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] [<IMPURITY>]
```

```
1dvt [ptype | ntype] [min.v=<EXPR>] [max.v=<EXPR>] \  
     [bias=<EXPR>] [bias.start] [bias.stop=<EXPR>] [bias.step=<EXPR>] \  
     [vb=<EXPR>] [temp.val=expr] [soi] [qss=<EXPR>] [workfunc=<EXPR>] \  
     [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>]
```

```
/* Default values : ptype, vb=0.0, qss=0 */
```

```
max.conc | min.conc [<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \  
                [region.occno=<EXPR>]
```

```
/* region.occno will default to all regions. */
```

```
2d.conc.file [<IMPURITY>] [<MATERIAL>] [mat.occno=<EXPR>] \  
            [x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR>]
```

```
max.conc.file | min.conc.file [<IMPURITY>] [<MATERIAL>] [xstep=<EXPR>] \  
                [x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR>]
```

```
max.bound | min.bound x.val=<EXPR> | y.val=<EXPR> \  
                [min.v=<EXPR>] [max.v=<EXPR>] [MATERIAL] [mat.occno=<EXPR>]
```

```

max.bound | min.bound x.pos | y.pos xval=<EXPR> y.val=<EXPR> \
[MATERIAL] [min.v=<EXPR>] [max.v=<EXPR>]

2d.area [<IMPURITY>] [x.step=<EXPR>] [min.v=<EXPR>] [max.v=<EXPR>] \
[x.max=<EXPR> x.min=<EXPR> y.max=<EXPR> y.min=<EXPR> |
y.max=<EXPR> y.min=<EXPR> region=<QSTRING>]

/* Default value : x.step = 10% of device size */

<CURVE_FUNC> (<CURVE_ARG>) :
  <CURVE_ARG>
  min (<CURVE_ARG>)
  /* Returns min y val for curve. */
  max (<CURVE_ARG>)
  /* Returns max y val for curve. */
  ave (<CURVE_ARG>)
  /* Returns average value for curve. */

slope | xintercept | yintercept (maxslope | minslope
(<CURVE_ARG>))

/* Takes the tangent to the curve with either the least or the
greatest slope and returns either the slope of this tangent,
or its x intercept, or its y intercept. */

area from (<CURVE_ARG>) [where x.min=<EXPR> and x.max=<EXPR>]

/* Determines the area under the specified curve between the x
limits defined by the min and max expressions. */

x.val from (<CURVE_ARG>) where y.val=<EXPR>
[and val.occno=<EXPR>]

y.val from (<CURVE_ARG>) where x.val=<EXPR>
[and val.occno=<EXPR>]

/* Determines the x (or y) ordinate on the curve where the
corresponding y (or x) ordinate is equal to the constant
expression for the occurrence specified. Linear interpolation
is used between points on the curve.*/

```

```
grad from (<CURVE_ARG>) where x.val=<EXPR> | y.val=<EXPR>
```

```
/* Determines the gradient at the first x (or y) ordinate on the curve
where the corresponding y (or x) value is equal to the consent expres-
sion. Linear interpolation is used between points on the curve.*/
```

```
<CURVE_SINGLE_LINE> :
```

```
curve (<AXIS_FUNC> (bias), \
      <AXIS_FUNC> (1dcapacitance [vg=<EXPR>] [vb=<EXPR>] \
        [bias.ramp=vg|vb] \
        [bias.step=<EXPR>] [bias.start=<EXPR>] \
        [bias.stop=<EXPR>] [temp.val=expr] [soi] \
        [qss=<EXPR>] \
        [workfunc=<EXPR>] \
        [x.val=<EXPR> | y.val=<EXPR> | \
        region=<QSTRING>] \
      ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
    )
```

```
/* Default values : vg=0.0, vb=0.0, bias.ramp=vg, qss=0 */
```

```
curve (<AXIS_FUNC> (depth), \
      <AXIS_FUNC> ([<IMPURITY>] [<MATERIAL>] \
        [mat.occno=<EXPR>] \
        [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] \
      ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
    )
```

```
curve (<AXIS_FUNC> (depth), \
```

```

<AXIS_FUNC> (srp
    [material="silicon"|"polysilicon"] [mat.occno=<EXPR>] \
    [x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING>] \
) \
[, xmin=<EXPR> xmax=<EXPR>] \
)

curve (<AXIS_FUNC> (<DEV_AXIS>), \
    <AXIS_FUNC> (<DEV_AXIS>) \
    [, xmin=<EXPR> xmax=<EXPR>] \
)

deriv (<AXIS_FUNC> (<DEV_AXIS>), \
    <AXIS_FUNC> (<DEV_AXIS>) \
    [, <INTEGER>] \
)

/* The integer is of course the nth derivative and its default value is
1.*/

edcurve (<DEFOCUS_AXIS>, <CRITICAL_DIMENSION_AXIS>, <DOSE_AXIS>, \
    dev=<EXPR> datum=<EXPR> x.step=<EXPR>)

<AXIS_FUNC> (<AXIS_ARG>) :

    <AXIS_ARG>

    <AXIS_ARG> + <EXPR>
    <EXPR>      + <AXIS_ARG>
    <AXIS_ARG> + <AXIS_ARG>

```



```
<AXIS_ARG> - <EXPR>
<EXPR>      - <AXIS_ARG>
<AXIS_ARG> - <AXIS_ARG>
```

```
<AXIS_ARG> / <EXPR>
<EXPR>      / <AXIS_ARG>
<AXIS_ARG> / <AXIS_ARG>
```

```
<AXIS_ARG> * <EXPR>
<EXPR>      * <AXIS_ARG>
<AXIS_ARG> * <AXIS_ARG>
```

```
<AXIS_ARG> ^ <EXPR>
<EXPR>      ^ <AXIS_ARG>
<AXIS_ARG> ^ <AXIS_ARG>
```

```
-<AXIS_ARG>
```

```
abs (<AXIS_ARG>)
log (<AXIS_ARG>)
log10 (<AXIS_ARG>)
sqrt (<AXIS_ARG>)
atan (<AXIS_ARG>)
```

```
<EXPR> :
```

```
<NUMBER>
```

```
$variable | $"variable"
```

```
/* deckbuild set variable, see section 5.8.2: Variable Substitution */
```

```
expr + expr
expr - expr
```

```

expr / expr
expr * expr

```

```

(expr)
-expr

```

```
<EXTRACT_MULTIPLE_LINE_SETUP_1> :
```

```
<EXTRACT_MULTIPLE_LINE_SETUP_A>
```

```
<EXTRACT_MULTIPLE_LINE_DONE_1> :
```

```

<CURVE_FUNC> (<CURVE_MULTIPLE_LINE_1>)
               [outfile=<QSTRING> [sigfigs=<EXPR>]]

```

```
<CURVE_MULTIPLE_LINE_1> :
```

```

curve (<AXIS_FUNC> (bias),                                     \
       <AXIS_FUNC> (1djunc.cap [<MATERIAL>] [mat.occno=<EXPR>] \
       [region.occno=<EXPR>] [junc.occno=<EXPR>]               \
       [temp.val=<EXPR>] [soi] [qss=<EXPR>] [workfunc=<EXPR>]   \
       [y.val=<EXPR>|x.val=<EXPR>|region=<QSTRING>]             \
       )                                                       \
       [, xmin=<EXPR> xmax=<EXPR>]                             \
       )

```

```
<EXTRACT_MULTIPLE_LINE_SETUP_2> :
```

```
<EXTRACT_MULTIPLE_LINE_SETUP_A>
```

```
<EXTRACT_MULTIPLE_LINE_DONE_2> :
```

```
<CURVE_FUNC> (<CURVE_MULTIPLE_LINE_2>)
```

```

[outfile=<QSTRING> [sigfigs=<EXPR>]]

<CURVE_MULTIPLE_LINE_2>:

curve (<AXIS_FUNC> (bias), \
      <AXIS_FUNC> ([p.ion | n.ion] [<MATERIAL>] \
                  [mat.occno=<EXPR>] \
                  [region.occno=<EXPR>] \
                  [junc.occno=<EXPR>] \
                  [temp.val=<EXPR>] [soi] [qss=<EXPR>] \
                  [workfunc=<EXPR>] \
                  [y.val=<EXPR> | x.val=<EXPR> | \
                  region=<QSTRING>] \
                  [an1=<EXPR>] [an2=<EXPR>] \
                  [bn1=<EXPR>] [bn2=<EXPR>] \
                  [ap1=<EXPR>] [ap2=<EXPR>] \
                  [bp1=<EXPR>] [bp2=<EXPR>] \
                  [betan=<EXPR>] [betap=<EXPR>] \
                  [egran=<EXPR>] \
                  ) \
      [, xmin=<EXPR> xmax=<EXPR>] \
      )

/* Default value : p.ion

an1=2.03e5, an2=7.03e5, bn1=1.231e6,
bn2=1.231e6,
ap1=6.71e5, ap2=1.582e6, bp1=1.693e6,
bp2=2.036e6,

betan=1.0, betap=1.0, egran=4e5

See Appendix A5: Threshold Voltage Calculation. */

<EXTRACT_MULTIPLE_LINE_SETUP_3> :

<EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

```

```

<EXTRACT_MULTIPLE_LINE_DONE_3> :

    <CURVE_FUNC> (<CURVE_MULTIPLE_LINE_3>)
                    [outfile=<QSTRING> [sigfigs=<EXPR>]]

<CURVE_MULTIPLE_LINE_3> :

    curve (<AXIS_FUNC> (bias), \
            <AXIS_FUNC> (1dsheet.res | 1dp.sheet.res | \
                        1dn.sheet.res | 1dconduct | \
                        1dp.conduct | 1dn.conduct \
                        [material="silicon" | "polysilicon"] \
                        [region.occno=<EXPR>] [mat.occno=<EXPR>] \
                        [y.val=<EXPR> | x.val=<EXPR> | \
                        region=<QSTRING>] \
                        [workfunc=<EXPR>] [soi] [semi.poly] \
                        [incomplete] \
                        [temp.val=<EXPR>] \
                        ) \
            [, xmin=<EXPR> xmax=<EXPR>] \
    )

<EXTRACT_MULTIPLE_LINE_SETUP_4> :

    <EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

<EXTRACT_MULTIPLE_LINE_DONE_4> :

    <CURVE_FUNC> (<CURVE_MULTIPLE_LINE_4>)
                    [outfile=<QSTRING> [sigfigs=<EXPR>]]

<CURVE_MULTIPLE_LINE_4> :

    curve (<AXIS_FUNC> (bias), \
            <AXIS_FUNC> (n.conc | p.conc | n.qfl \
                        |p.qfl | intrinsic | potential
    )

```

```

        | n.mobility | p.mobility
        | efield     | econductivity \
[material="silicon" | "polysilicon"] \
    [region.occno=<EXPR>] [mat.occno=<EXPR>] \
[y.val=<EXPR> | x.val=<EXPR> |
region=<QSTRING>] \
[workfunc=<EXPR>] [soi] [semi.poly]
[temp.val=<EXPR>] \
    ) \
    [, xmin=<EXPR> xmax=<EXPR>] \
    )
<EXTRACT_MULTIPLE_LINE_DONE_5> :

    sheet.res|p.sheet.res|n.sheet.res|conduct|p.conduct|n.conduct \
[material="silicon"|"polysilicon"] [region.occno=<EXPR>] \
[mat.occno=<EXPR>] \
[y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>] \
[workfunc=<EXPR>] [soi] [semi.poly] [incomplete]
[temp.val=<EXPR>]

<EXTRACT_MULTIPLE_LINE_SETUP_5> :
    <EXTRACT_MULTIPLE_LINE_SETUP_A> | <EXTRACT_MULTIPLE_LINE_SETUP_B>

<EXTRACT_MULTIPLE_LINE_SETUP_A> :

    [<MATERIAL>] [mat.occno=<EXPR>] [region.occno=<EXPR>] \
[bias=<EXPR>] [bias.start=<EXPR>] [bias.step=<EXPR>] \
[bias.stop=<EXPR>] \
[y.val=<EXPR> | x.val=<EXPR> | region=<QSTRING>]

<EXTRACT_MULTIPLE_LINE_SETUP_B> :

    [interface.occno=<EXPR>] [qss=<EXPR>]

    /* Default value : qss=1e10 */

```

```
<DEV_AXIS> :

v."<electrode>"          /* voltage at electrode */

i."<electrode>"          /* current at electrode

c."<electrode1>" "<electrode2>" /* capacitance between
                               electrode1 and electrode2 */

g."<electrode1>" "<electrode2>" /* conductance between
                               electrode1 and electrode2 */

vint."<electrode>"      /* internal voltage at
                               electrode */

time                    /* transient time */

temperature | temp     /* device temperature */

frequency | freq       /* frequency */

beam."<beam no>"        /* light intensity for specified
                               beam */

ie."<electrode>"        /* electron current at
                               electrode */

ih."<electrode>"        /* hole current at electrode */

q."<electrode>"         /* charge at electrode */

id."<electrode>"        /* displacement current at
                               electrode */

ireal."<electrode>"    /* real component of current at
                               electrode */

iimag."<electrode>"    /* imaginary component of current
                               at electrode */
```

```
ifn."<electrode>"      /* fowler nordhiem current at
                        electrode */

ihe."<electrode>"      /* hot electron current at
                        electrode */

ihh."<electrode>"      /* hot hole current at
                        electrode */

wfd."<electrode>"      /* workfunction difference at
                        electrode */

rl."<electrode>"       /* lumped resistance at
                        electrode */

cl."<electrode>"       /* lumped capacitance at
                        electrode */

ll."<electrode>"       /* lumped inductance at
                        electrode */

vcct.node."<circuit node>" /* circuit bias */

icct.node."<circuit node>" /* circuit current */

rhoe."<layer>"         /* Electron sheet resistance for
                        specified layer */

rhoh."<layer>"         /* Hole sheet resistance for
                        specified layer */

rho."<layer>"          /* Total sheet resistance for
                        specified layer */

vlayer."<layer>"       /* Bias on specified layer */

sm."<mode>"           /* Photon density for specified
                        mode */

pm."<mode>"           /* Laser power per mirror for
                        specified mode */
```

```
gm."<mode>"                /* Gain for specified mode */

vcct.real."<circuit node>"  /* Real circuit bias */

vcct.imag."<circuit node>" /* Imaginary circuit bias */

icct.real."<circuit node>" /* Real circuit current */

icct.imag."<circuit node>" /* Imaginary circuit current */

s.real."<parameter>" - real value of specified "S" parameter

s.imag."<parameter>" - imaginary value of specified "S"
                    parameter

y.real."<parameter>" - real value of specified "Y" parameter

y.imag."<parameter>" - imaginary value of specified "Y"
                    parameter

z.real."<parameter>" - real value of specified "Z" parameter

z.imag."<parameter>" - imaginary value of specified "Z"
                    parameter

h.real."<parameter>" - real value of specified "H" parameter

h.imag."<parameter>" - imaginary value of specified "H"
                    parameter

abcd.real."<parameter>" - real value of specified "ABCD"
                    parameter

abcd.imag."<parameter>" - imaginary value of specified "ABCD"
                    parameter

probe."<probe name>"      /* Atlas probe values */
```



```

ow. "beam no"                /*optical wavelength
                             for specified beam */

spc. "beam no"                /*source photo current
                             for specified beam */

apc. "beam no"                /*available photo current
                             for specified beam */

elect."<PARAMETER>"          /* Value for specified
                             electrical parameter */

<PARAMETER> :                /* ALL PARAMETER strings
                             are completely case
                             -insensitive. */

Time
Freq
Frequency
Temp
Temperature

Absorption
Absorption Coefficient
Absorption Spectrum
Acceptor bump state density
Acceptor Bump State f_T
Acceptor Bump State Ionized Density
Acceptor Int. Bump State Density
Acceptor Int. Bump State f_T
Acceptor Int. Bump State Ionized Density
Acceptor Int. Tail State Density
Acceptor Int. Tail State f_T
Acceptor Int. Tail State Ionized Density
Acceptor Int. Trap Density
Acceptor Int. Trap f_T
Acceptor Int. Trap Ionized Density
Acceptor state energy
Acceptor tail state density
Acceptor Tail State f_T
Acceptor Tail State Ionized Density

```

Acceptor Trap Density
Acceptor Trap f_T
ADF
Amphoteric Int. Trap (0) State Density
Amphoteric Int. Trap (-) State Density
Amphoteric Int. Trap (+) State Density
Amphoteric Int. Trap Total State Density
Amphoteric Trap (0) f_T
Amphoteric Trap (0) State Density
Amphoteric Trap (-) f_T
Amphoteric Trap (+) f_T
Amphoteric Trap (-) State Density
Amphoteric Trap (+) State Density
Amphoteric Trap Total State Density
Angle
Anneal Temperature
Available photo current
Average Annual Power Production
Average Relative Permittivity
Channel Sheet Conductance
CIE X Coordinate
CIE Y Coordinate
CIE Z Coordinate
Computation time
Conversion Efficiency
Current gain
Cutoff frequency
Displacement current
Distance along line
Donor bump state density
Donor Bump State f_T
Donor Bump State Ionized Density
Donor Int. Bump State Density
Donor Int. Bump State f_T
Donor Int. Bump State Ionized Density
Donor Int. Tail State Density
Donor Int. Tail State f_T
Donor Int. Tail State Ionized Density

Donor Int. Trap Density
Donor Int. Trap f_T
Donor Int. Trap Ionized Density
Donor state energy
Donor tail state density
Donor Tail State f_T
Donor Tail State Ionized Density
Donor Trap Density
Donor Trap f_T
Effective refractive index (Im)
Effective refractive index (Re)
Electric field
Electron Conc
Electron current
Electron effective mass
Electron energy
Electron energy relax time
Electron Impact Ionization Rate
Electron kinetic energy
Electron mass
Electron mobility
Electron momentum relax time
Electron Peltier Coefficient
Electron potential energy
Electron temp
Electron Velocity
Energy
e- Quantum Well Capture Rate
Function 1
Function 2
Gain
Gain (TE)
Gain (TM)
Gamma valley ratio
Generation rate
Global device temperature
Gma
Gms

Haze Parameter Cr
Haze Parameter Ct
Heavy hole valley ratio
Hole Conc
Hole current
Hole effective mass
Hole energy
Hole energy relaxation time
Hole Impact Ionization Rate
Hole kinetic energy
Hole mass
Hole mobility
Hole momentum relaxation time
Hole Peltier Coefficient
Hole potential energy
Hole temp
Hole velocity
h+ Quantum Well Capture Rate
<I1.I1*>
<I2.I2*>
Imag(<I1.I2*>)
Imaginary Index
Imag(<V1.V2*>)
Imag(Zo)
Integrated e- Conc
Integrated h+ Conc
Ionized Acceptor Concentration
Ionized Acceptor Trap Concentration
Ionized Donor Concentration
Ionized Donor Trap Concentration
Iteration
Lateral Field Gradient
Lattice temp
Light Frequency
Light hole valley ratio
Luminescent power
Luminescent wavelength
Luminous Efficiency

Luminous Flux
Luminous Intensity
L valley ratio
Max transducer power gain
MC number of samples
Minimum noise figure
Net doping
Noise conductance
Norm Intensity
Optical Output Coupling
Optical source frequency
Optical wavelength
output current average deviation
output current standard deviation
Output Spectral Power Density
Photon Density
Photon Energy
PL Intensity
Polarization
Position X
Position Y
Position Z
Potential
Prob. of Avalanche by Elec.
Prob. of Avalanche by Hole
Prob. of Avalanche Joint
Propagation Angle Phi
Propagation Angle Theta
Quantum Efficiency
Quantum Temperature
Radiative Rate
Real (<I1.I2*>)
Real Index
Real (<V1.V2*>)
Real (Zo)
Reciprocal Electric Field
Recombination rate
Reflective Haze

Reflectivity
Relative permittivity
Resistivity
RMS Error
SEU Track Charge
SEU Track Cumulative Charge
SEU Track Generation Rate
Sheet Capacitance
Sheet Electron Density
simulation time
Small signal frequency
SONOS Blocking Insulator Current
SONOS Tunnelling Insulator Current
Source photo current
Spin-orbit hole valley ratio
Spontaneous emission rate
Stern stability factor
Surface Field
Surface Potential
TE Radiant Intensity
Threshold Voltage
Time step magnitude
Time step number
TM Radiant Intensity
Total Acceptor Int. Trap Density
Total Acceptor Int. Trap Ionized Density
Total acceptor trap density
Total Acceptor Trap Ionized Density
Total Donor Int. Trap Density
Total Donor Int. Trap Ionized Density
Total donor trap density
Total Donor Trap Ionized Density
Total integration time
Total Optical Power
Total Photon Flux
Total Power Emitted
Total Power per Mirror
Total QuasiStatic C-V

Total Radiant Intensity
Total Radiation Dose
Total Trap State Density
Transient time
Transmission
Transmissive Haze
Trapped Interface Electron Charge
Trapped Interface Hole Charge
Unilateral power gain
UTMOST Measurement
UTMOST Measurement (Linear)
UTMOST Measurement (Log)
<V1.V1*>
<V2.V2*>
Victory AMR buffwidth
Victory AMR Level
Victory AMR mBulkDist
Victory AMR mDeltMax
Victory AMR mDeltMin
Victory AMR mDeltX
Victory AMR mDeltY
Victory AMR mDeltZ
Victory AMR mesh id
Victory AMR mesh location
Victory AMR mesh nx
Victory AMR mesh ny
Victory AMR mesh nz
Victory AMR mesh region
Victory AMR mesh region distrace/Refine flag
Victory AMR mesh sx
Victory AMR mesh sy
Victory AMR mesh sz
Victory AMR mLayer
Victory AMR mLayerHistory
Victory AMR mMaterialFlag
Victory AMR refine
Victory AMR regrid
VPICM Regular Node Index

Wavelength
X valley ratio

<DEFOCUS_AXIS> :

da.value"DEFOCUS" | da.value"<CURVE_NUMBER>" "DEFOCUS"

<CRITICAL_DIMENSION_AXIS> :

da.value"CDS" | da.value"<CURVE_NUMBER>" "CDS"

<DOSE_AXIS> :

da.value"DOSE" | da.value"<CURVE_NUMBER>" "DOSE"

<CURVE_NUMBER> :

/* Integer specifying which curve when multiple curves are
present in a DA format file. */

<MATERIAL> : /* All MATERIAL strings are completely
case -insensitive. */

3C-SiC
4H-SiC
6H-SiC
Air
Al2O3
AlAs
AlAsSb
AlGaAs
AlGaAsP
AlGaAsSb
AlGaN
AlGaNAs
AlGaNp
AlGaP
AlGaSb

AlInAs
AlInNAs
AlInNP
AlN
AlP
AlPAs
Alpha Si 1
Alpha Si 2
Alpha Si 3
Alpha Si 4
AlPSb
Alq3
AlSb
AlSi
AlSiCu
AlSiTi
AlSix
Aluminum
AlxGa1_xAs_x_0.25
AlxGa1_xAs_x_0.5
AlxGa1_xAs_x_0.75
AlxIn1_xAs_x_0.50
Ambient
Ba2NdCu3O7
Ba2YCu3O7
BAlq
Barrier
BeTe
BPSG
BSG
CBP
CdO
CdS
CdSe
CdTe
CdZnO
CdZnTe
CMO

CNT
Cobalt
Computational Window
Conductor
Contact
Cooling package material
Copper
CoSix
CuInGaSe
CuPc
Diamond
Fictive GaAs
GaAs
GaAsP
GaAsSb
GaN
GaP
Gas
GaSb
GaSbAs
GaSbP
GaSbTe
Germanium
Gold
GST
HfO2
HfSiO4
HgCdTe
HgS
HgSe
HgTe
IGZO
IMO
In2O3
InAlAs
InAlAsP
InAlAsSb
InAlGaAs

InAlGaN
InAlGaP
InAlN
InAlP
InAlSb
InAs
InAsP
InAsSb
InGaAs
InGaAsP
InGaAsSb
InGaN
InGaNAs
InGaNp
InGaP
InGaSb
InN
InP
InPAsSb
InPSb
InSb
Insulator
InxGal_xAs_x_0.33 Str GaAs
InxGal_xAs_x_0.50 Unstr
InxGal_xAs_x_0.75 Str InP
Iron
Irppy
ITO
Lead
Lens
Liquid Crystal
Mask Clear
Mask Opaque
MgCdO
MgCdZnO
MgO
MgZnO
Molybdenum

MoSix
Nickel
NiSix
NPB
NPD
Organic
OxyNitride
Palladium
PbS
PbSe
PbTe
PCM
PdSix
Pentacene
Phase Shift
Photoresist
Platinum
PM
PMMA
Polyimide
Polymer
Polysilicon
PPV
PtSix
Sapphire
ScN
Se
Si
Si~3N~4
SiGe
Silicon
Silver
SiN
SiO~2
SixNyHz
SnO2
SnTe
SOG

Tantalum
TaSix
TEOS
Tetracene
Tin
TiNi
TiO
TiO2
TiON
TiSix
Titanium
TiW
TPD
Tungsten
Twisted Nematic Liquid Crystal
Vacuum
WSix
ZnO
ZnS
ZnSe
ZnTe
ZrO2
ZrSix
material=<QSTRING>

<IMPURITY> : /* All IMPURITY strings are completely
case -insensitive. */

Absorption Coefficient
Acceptor Conc
Acceptor Trap Concentration
Acceptor Trap Density
Active Aluminum
Active Antimony
Active Arsenic
Active Beryllium
Active Boron
Active Carbon
Active Chromium

Active Fluorine
Active Gallium
Active Germanium
Active Gold
Active Helium
Active Hydrogen
Active Indium
Active Magnesium
Active Nitrogen
Active Oxygen
Active Phosphorus
Active Selenium
Active Si-28
Active Si-29
Active Tin
Active Zinc
alpha
Aluminum
Antimony
Applied Potential
Arsenic
As active poly grain boundary
As active poly grain interior
AsI pairs
As poly grain boundary
As poly grain interior
Atomic Hydrogen Conc
Auger Recomb Eff Lifetime
Auger Recomb Rate
Average e- Energy
Average E Field X
Average E Field Y
Average E Field Z
Average Electrons Conc
Average e- Velocity X
Average e- Velocity Y
Average e- Velocity Z
Average Generation Rate

Average h+ Energy
Average Holes Conc
Average h+ Velocity X
Average h+ Velocity Y
Average h+ Velocity Z
Average Impact Gen Rate
Average Potential
Average Relative Permittivity
B active poly grain boundary
B active poly grain interior
Band to Band Tunnel Current Density
Band to Band Tunneling Factor
Band to Band Tunneling Rate
Beryllium
BI pairs
Boron
Bound State Electron Concentration
Bound State Hole Concentration
B poly grain boundary
B poly grain interior
CAR acid conc
Carbon
Carrier Conc
Celsius Temp
Charge Conc
Chromium
Ci*
Composition fraction Z
Composition X
Composition Y
Cond. Current Density
Cond Current X
Cond Current Y
Cond Current Z
Conduction Band Energy
Cooling Package Temp
Current Flowlines
Cv*

Delta Area
Dislocation Loop Conc
Dislocation Loop Size
Disp. Current Density
Displ Current X
Displ Current Y
Displ Current Z
Donor Conc
Donor Trap Concentration
Donor Trap Density
Dopant Langevin Recomb Rate
Dopant Singlet Exciton Density
Dopant Triplet Exciton Density
Dry O~2
dT/dx
dT/dy
dT/dz
D Vector Magnitude
D Vector X
D Vector Y
D Vector Z
Ec (T)
e- Current Density
Eff BGN w/o T-depend
Effective BGN
Effective BGN (C-band)
Effective BGN (V-band)
Effective nie
Eff Min Carr Lifetime
Eff. Refractive Index (Im)
Eff. Refractive Index (Re)
E Field X
E Field Y
E Field Z
Eg
e- Ground State Energy
Eg (T)
Einstein Rel Corr e-

Einstein Rel Corr h+

e- Ionization Coefficient

e- Ioniz Eff Field

Electrical Conductivity

Electric Field

Electrode #

Electron Affinity

Electron Conc

Electron Concentration Update

Electron Conc (linear)

Electron Continuity Equation RHS

Electron Diff Coeff

Electron effective mass

Electron energy

Electron energy relax time

Electron kinetic energy

Electron mass

Electron momentum relax time

Electron potential energy

Electron QFL

Electron Quantum Potential

Electron Reaction Rate

Electrons

Electron Temperature

Electron Temperature Equation RHS

Electron Temperature Update

Electron Velocity

Electrostatic Potential Update

e- Mobility

e- Mobility Lateral

e- Mobility Trans

e- Mobility X

e- Mobility Y

e- Mobility Z

e- Nonlocal BBT Current Density

e- Peltier Coeff

e- Quantum Well Capture Rate

Equilibrium Electron Conc

Equilibrium Hole Conc
Equilibrium Potential
e- Recomb Rate
e- SRH Tno
e- Thermal Velocity
e- Thermoelectric Power
e- Tunnel Current Density
e- Velocity X
e- Velocity Y
e- Velocity Z
Ev (T)
Ex
Exciton Dissociation Efficiency
Exciton Dissociation Rate
Ex (envelope)
Ey
Ey (envelope)
Ez
Ez (envelope)
Fast State Density
Fixed Oxide Charge
Fluorine
Free Carrier Loss
Gallium
Gamma valley ratio
Germanium
Gold
h+ Current Density
Heat Capacitance
Heat Capacity
Heat Conductance
Heat Conductivity
Heat Flow Density X
Heat Flow Density Y
Heat Flow Density Z
Heat Flow Equation RHS
Heavy hole valley ratio
Helium

h+ Ground State Energy
h+ Ionization Coefficient
h+ Ioniz Eff Field
h+ Mobility
h+ Mobility Lateral
h+ Mobility Trans
h+ Mobility X
h+ Mobility Y
h+ Mobility Z
h+ Nonlocal BBT Current Density
Hole Conc
Hole Concentration Update
Hole Conc (linear)
Hole Continuity Equation RHS
Hole Diff Coeff
Hole effective mass
Hole energy
Hole energy relaxation time
Hole kinetic energy
Hole mass
Hole momentum relaxation time
Hole potential energy
Hole QFL
Hole Quantum Potential
Hole Reaction Rate
Holes
Hole Temperature
Hole Temperature Equation RHS
Hole Temperature Update
Hole Velocity
Hot Electron Current Density
Hot Hole Current Density
h+ Peltier Coeff
h+ Quantum Well Capture Rate
h+ Recomb Rate
h+ SRH Tno
h+ Thermal Velocity
h+ Thermoelectric Power

h+ Tunnel Current Density
h+ Velocity X
h+ Velocity Y
h+ Velocity Z
Hx
Hx (envelope)
Hy
Hydrogen Conc
Hydrostatic Pressure
Hy (envelope)
Hz
Hz (envelope)
I
Idiffuse
ILAV Node Flags
Imag(LNS<V1.V2*>)
Impact Gen'd Carriers
Impact Gen Rate
Implant Damage
In active poly grain boundary
In active poly grain interior
Indium
In poly grain boundary
In poly grain interior
Insulator Charge
Intensity
Interface Charge
Interstitial Arsenic
Interstitial Clusters
Interstitial Gallium
Interstitials
Intrinsic Conc (nie)
Ionic Species 1 Conc
Ionic Species 2 Conc
Ionic Species 3 Conc
Ionization Coeff e-
Ionization Coeff h+
Ionization Effect Field

Ionized Acceptor Concentration
Ionized Acceptor Trap Concentration
Ionized Donor Concentration
Ionized Donor Trap Concentration
Ispicular
J (diff.)
J (drift)
Je- X
Je- Y
Je- Z
Jh+ X
Jh+ Y
Jh+ Z
Jn (diff.)
Jn (drift)
Jnx (diff.)
Jnx (drift)
Jny (diff.)
Jny (drift)
Jnz (diff.)
Jnz (drift)
Joi X
Joi Y
Joi Z
Joule Heat Power
Jp (diff.)
Jp (drift)
Jproton X
Jproton Y
Jproton Z
Jpx (diff.)
Jpx (drift)
Jpy (diff.)
Jpy (drift)
Jpz (diff.)
Jpz (drift)
Jsx
Jsy

Jsz
Jtot X
Jtot Y
Jtot Z
Jvo X
Jvo Y
Jvo Z
Jx (diff.)
Jx (drift)
Jy (diff.)
Jy (drift)
Jz (diff.)
Jz (drift)
KSN
KSP
Langevin Recomb Rate
Lateral Field Gradient
Lattice Temperature
Lattice Temperature Update
Light hole valley ratio
Light intensity, mode 1
Light intensity, mode 2
Light intensity, mode 3
Light intensity, mode 4
Light intensity, mode 5
Light intensity, mode 6
Light intensity, mode 7
Light intensity, mode 8
Light intensity, mode 9
Light intensity, mode 10
Linear Charge Conc
Linear Net Doping
Linear Total Doping
LNS <V1.V1*>
LNS <V2.V2*>
Local Optical Gain
L valley ratio
Magnesium

Material Density
Material Type #
Mat Type # w/o Electrodes
Maxwell Stress
Maxwell Stress X
Maxwell Stress Y
Maxwell Stress Z
Mesh Discretization Error Estimate
Metal Density
Molecular Hydrogen Conc
Msx
Msy
Msz
Nc
Nc (T)
Net Active Doping
Net Doping
Ni
N Int X
N Int Y
N int Z
Nitride Electron Charging Rate
Nitride Electron Recombination Rate
Nitride Hole Charging Rate
Nitride Hole Recombination Rate
Nitrogen
n-MobilityEnhancement XX
n-MobilityEnhancement XY
n-MobilityEnhancement XZ
n-MobilityEnhancement YY
n-MobilityEnhancement YZ
n-MobilityEnhancement ZZ
Node Index
Nonlocal BBT e- Tunnelling Rate
Nonlocal BBT h+ Tunnelling Rate
Nonlocal e- Tunnelling Rate
Nonlocal h+ Tunnelling Rate
Nonlocal TAT e- Gamma

Nonlocal TAT h+ Gamma
Norm Grad Int
Norm Intensity
(n.p)^{1/2} (T)
Nv
Nv (T)
nxx
nyy
nzz
Occupancy Trap #1
Occupancy Trap #2
Occupancy Trap #3
Occupancy Trap #4
Occupancy Trap #5
Occupancy Trap #6
Occupancy Trap #7
Occupancy Trap #8
Occupancy Trap #9
Occupancy Trap #10
Oi Velocity
Oi Velocity Enhancement
Oi Velocity Enhancement X
Oi Velocity Enhancement Y
Oi Velocity Enhancement Z
Oi Velocity X
Oi Velocity Y
Oi Velocity Z
Optical Intensity
output current average deviation
output current standard deviation
Oxygen
Oxygen Conduction Band Energy
Oxygen Diff Coeff
Oxygen Exchange Recombination Rate
Oxygen Interstitial Conc
Oxygen Interstitial Current Density
Oxygen Interstitial QFL
Oxygen QFL

Oxygen Recombination Rate
Oxygen Vacancy Conc
Oxygen Vacancy Current Density
Oxygen Vacancy QFL
Oxygen Valence Band Energy
PAC
Package Layer #
Package Material #
Peltier-Thomson Heat Power
Permeability
Ph active poly grain boundary
Ph active poly grain interior
Phosphorus
Photogeneration Rate
Photogeneration Rate (linear)
Photon Absorption Rate
Ph poly grain boundary
Ph poly grain interior
P Int X
P Int Y
P Int Z
PI pairs
p-MobilityEnhancement XX
p-MobilityEnhancement XY
p-MobilityEnhancement XZ
p-MobilityEnhancement YY
p-MobilityEnhancement YZ
p-MobilityEnhancement ZZ
Point Index
Poisson Equation RHS
Polarization Charge Conc
Poly Grain Size
Poly Grain Size X
Poly Grain Size Y
Poly Grain Size Z
Potential
Potential (process)
Prob. of Avalanche by Elec.

Prob. of Avalanche by Hole
Prob. of Avalanche Joint
Proton Conc
Proton Current Density
QFL Gradient X
QFL Gradient Y
QFL Gradient Z
Quantum Potential
Radiative Recomb Rate
Ratio NIE/Maj Carr Conc
Ratio nie/ni
Real(LNS<V1.V2*>)
Rec Heat Power
Recombination Rate
Refractive index (Im)
Refractive Index (Re)
Region #
Region # w/o Electrodes
Relative Permittivity
Resist Elevation
Resistivity
Rstim, mode 1
Rstim, mode 2
Rstim, mode 3
Rstim, mode 4
Rstim, mode 5
Rstim, mode 6
Rstim, mode 7
Rstim, mode 8
Rstim, mode 9
Rstim, mode 10
Sb active poly grain boundary
Sb active poly grain interior
SbI pairs
Sb poly grain boundary
Sb poly grain interior
Selenium
Semi Fixed Charge

Sem/Ins Material #
Sem/Ins # w/o Electrodes
Shear Stress
Sheet Capacitance
Sheet Electron Density
Sigma star x
Sigma star y
Sigma star z
Sigma x
Sigma y
Sigma z
Silicon
Silicon-28
simulation time (s)
Singlet Exciton Density
Singlet Exciton Generation Rate
Slow State Density
Species 1 Reaction Rate
Species 2 Reaction Rate
Species 3 Reaction Rate
Spin-orbit hole valley ratio
SRH par n_1/n_2
SRH Rec Eff Lifetime
SRH Recomb Rate
SRP Doping
Stimulated Recombination Rate
Stokes Vector I
Stokes Vector Q
Stokes Vector U
Stokes Vector V
Strain XX
Strain XY
Strain XZ
Strain YY
Strain YZ
Strain ZZ
Stress XX
Stress XY

Stress XZ
Stress YY
Stress YZ
Stress ZZ
Structure Temp
Surface Field
Surface Potential
Surface Recombination
Surf Recomb Eff Lifetime
Te Gradient
Th Gradient
Thomson Heat Power
Tin
Total Current Density
Total Doping
Total Field
Total Heat Flow Density
Total Heat Power
Total T Gradient
Trap Density #1
Trap Density #2
Trap Density #3
Trap Density #4
Trap Density #5
Trap Density #6
Trap Density #7
Trap Density #8
Trap Density #9
Trap Density #10
Trap Electron Capture Rate
Trap Hole Capture Rate
Trapped e- Density (Acc)
Trapped h+ Density (Dnr)
Trapped Insulator e- Concentration
Trapped Insulator h+ Concentration
Traps
Trap State Recomb
Triplet Exciton Density

```
Triplet Exciton Generation Rate
Tunnelling Rate
User Recomb Eff Lifetime
User Recomb Rate
Vacancies
Vacancy Arsenic
Vacancy Gallium
Valence Band Energy
Velocity X
Velocity Y
Velocity Z
Viscosity
von Mises Stress
Vo Velocity
Vo Velocity Enhancement
Vo Velocity Enhancement X
Vo Velocity Enhancement Y
Vo Velocity Enhancement Z
Vo Velocity X
Vo Velocity Y
Vo Velocity Z
Wet O~2
X Position
X valley ratio
Y Position
Zinc
Z Plane Index
Z Position
impurity=<QSTRING>
<NUMBER> :

    /* Real or integer value. */

<QSTRING> :

    /* Quoted string, for example, "silicon". */
```

5.3.2 DEFAULTS

The following default values will be assumed.

```
name=<QSTRING>           : name=None

<MATERIAL>               : material="silicon"

<IMPURITY>              : impurity="net doping"

mat.occno=<EXPR>         : mat.occno=1

junc.occno=<EXPR>       : junc.occno=1

region.occno=<EXPR>     : region.occno=1

interface.occno=<EXPR>  : interface.occno=1

val.occno=<EXPR>        : val.occno=1

datafile=<QSTRING>      : datafile="results.final"

outfile=<QSTRING>       : outfile="extract.dat"

bias.step=<EXPR>        : bias.step=0.25

bias.start=<EXPR>       : bias.start=0.0

bias.stop=<EXPR>        : bias.stop=5.0

temp.val=<EXPR>         : temp.val=300.0

soi                      : FALSE

semi.poly                : FALSE

incomplete               : FALSE

x.val=<EXPR> | y.val=<EXPR> | region=<QSTRING> : x.val is set to
                                                    be 5% from
```

left-hand side of
structure.

```
sigfigs=<EXPR> : sigfigs=6
```

5.3.3 Examples of Process Extraction

Note: You can enter `extract` commands on multiple lines using a backslash character for continuation. The syntax, however, shown below should be entered on a single line although shown on two or more lines.

The following examples assume to be extracting values from the current simulation running under DeckBuild. You can use saved standard structure files directly with `extract` using the syntax below.

```
extract init infile="filename"
```

Material Thickness

Extract the thickness of the top (first) occurrence of Silicon Oxide for a 1D cutline taken where $Y=0.1$ (Assume 2D structure). A warning is then displayed if results cross boundaries set by `max.v` and `min.v`.

```
extract name="tox" thickness material="SiO~2" mat.occno=1 y.val=0.1
min.v=100 max.v=500
```

oxide can be substituted for the `material="SiO~2"`.

Junction Depth

Extract the junction depth of the first junction occurrence in the top (first) occurrence of silicon for a 1D cutline taken where $X=0.1$.

```
extract name="j1 depth" xj material="Silicon" mat.occno=1 x.val=0.1
junc.occno=1
```

Surface Concentration

Extract the surface concentration net doping for the top (first) occurrence of silicon for a 1D cutline taken for an X value corresponding to the gate contact/region for loaded MaskViews cutline data.

```
extract name="surface conc" surf.conc impurity="Net Doping" material="Silicon" mat.occno=1 region="gate"
```

QUICKMOS 1D Vt

Extract the 1D threshold voltage of a p-type MOS cross section at $x=0.1$ using the built-in QUICKMOS 1D device simulator. This example uses a default gate bias setting of 0-5V for a 0.25V step with the substrate at 0V and a default device temperature of 300 Kelvin. Values of QSS and gate workfunction have also be specified.

```
extract name="1D Vt" 1dvt ptype qss=1e10 workfunc=5.09 x.val=0.1
```

This 1D Vt extraction will calculate the 1D threshold voltage of an n-type MOS cross section at $x=0.1$, where a gate voltage range (0.5-20V) was specified while the substrate (V_b) is set at 0.2V. The device temperature has been set to 350 Kelvin.

```
extract name="1D Vt 0-20v" 1dvt ntype bias=0.5 bias.step=0.25
bias.stop=20.0 vb=0.2 temp.val=350.0 x.val=0.1
```

Sheet Resistance and Sheet Conductance

Note: For sheet conductance extraction substitute "sheet.res" with "conduct" (e.g., conduct, p.conduct, n.conduct).

Extract the total sheet resistance of the first p-n region in the top (first) occurrence of polysilicon for a cutline at $x=0.1$. Polysilicon is treated as a metal by default but is flagged here as a semiconductor (`semi.poly`). The default device temperature of 300 Kelvin and no layer biases will be used and the incomplete ionization flag is also set for carrier freezeout calculations (see "Incomplete Ionization Of Impurities" physics section within the Atlas manual).

```
extract name="Total SR" sheet.res material="Polysilicon" mat.occno=1
x.val=0.1 region.occno=1 semi.poly incomplete
```

Extract the n-type sheet resistance of the second p-n region in the top (first) occurrence of silicon for a cutline at $x=0.1$, where the second region is held at 4.0V and the device temperature is set to 325 Kelvin. These commands use the `start/cont/done` syntax to create a multi-line statement as described in [Section 5.8 "Extract Features"](#).

```
extract start material="Silicon" mat.occno=1 region.occno=2 bias=4.0
x.val=0.1 extract done name="N-type SR" n.sheet.res material="Sil-
icon" mat.occno=1 temp.val=325 x.val=0.1 region.occno=2
```

The following multi-line statement extracts the p-type sheet resistance of the first p-n region in the top (first) occurrence of silicon for a cutline at $x=0.1$, where the first region is held at 5.0V. The second region is held at 1.0V and the first interface Q_{ss} value equal to $1e10$.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=5.0
x.val=0.1
extract cont material="Silicon" mat.occno=1 region.occno=2 bias=1.0
x.val=0.1
extract cont interface.occno=1 qss=1.0e10
extract done name="P-type SR" p.sheet.res material="Silicon" mat.occno=1
x.val=0.1 region.occno=1
```

Note: This is an example of the multi-line "start/continue/done" type of statement used to specify layer biases and Q_{ss} values. It is recommended that you always let the Extract popup write this particular syntax. The Q_{ss} value also specifies the material interface occurrence involved, counting from the top down. There can be any number of additional "continue" lines to specify the biases on other layers and the Q_{ss} values of other interfaces; the last line, "done", does the actual extraction.

1D Max/Min Concentration

Extract the peak concentration of net doping within the first p-n region of the top (first) layer of silicon for a 1D cutline at $x=0.1$.


```
extract name="Max 1d Net conc" max.conc impurity="Net Doping"
material="Silicon" mat.occno=1 x.val=0.1 region.occno=1
```

Extract the peak concentration of phosphorus within any p-n regions (default) for all materials using a 1D cutline at $x=0.1$.

```
extract name="Max 1d phos conc" max.conc impurity="Phosphorus" mate-
rial="All" x.val=0.1
```

Extract the minimum concentration of boron within any p-n regions of the top (first) layer of silicon for a 1D cutline at $x=0.1$.

```
extract name="Min 1d bor conc" min.conc impurity="Boron" material="Sili-
con" mat.occno=1 x.val=0.1
```

2D Max/Min Concentration

Extract the peak concentration of net doping for the entire 2D structure.

```
extract name="Max 2D net conc" 2d.max.conc impurity="Net Doping" mate-
rial="All"
```

Extract the peak concentration of boron within the silicon material in the 2D “box” limits defined.

```
extract name="Max 2D bor conc" 2d.max.conc impurity="Boron" material="Sil-
icon" y.min=0.1 y.max=0.9 x.min=0.2 x.max=0.6
```

In addition to this statement, you can add the `interpolate` flag. When present, this flag causes the extraction to perform interpolation at the edges of the specified bounding box for min/max concentration and position.

Extract the minimum concentration of phosphorus for all materials within the 2D “box” limits. These limits are defined by user-defined y coordinates and x values corresponding to loaded MaskViews outline information for the specified electrode or region.

```
extract name="Min 2D phos conc" 2d.min.conc impurity="Phosphorus" mate-
rial="All" region="gate" y.min=0.1 y.max=0.9
```

The following multi-line extract command measures the minimum concentration of antimony for the entire 2D structure and return the x - y coordinates of the extracted concentration.

```
extract name="Min 2D ant conc" 2d.min.conc impurity="Antimony" mate-
rial="All"
extract name="Min 2D ant conc X position" x.pos
extract name="Min 2D ant conc Y position" y.pos
```

Note: The x - y position syntax must directly follow the 2D concentration extraction (same as `start/continue/done` syntax). We advise you to use the Extract popup to create these statements.

2D Concentration File

The output file contains data of the format $x\ y\ c$, where c is the value of concentration at the coordinates xy . The following example extracts the boron concentration in Silicon for the whole structure.

```
extract 2d.conc.file material="silicon" impurity="boron"
outfile="conc.dat"
```

You can increase the precision of the values written to the output file using the `sigfigs` option.

```
extract 2d.conc.file material="silicon" impurity="boron"
outfile="conc.dat" sigfigs=10
```

The number of significant figures defaults to 6.

1D Material Region Boundary

Extracting the maximum Y boundary (upper side) location of the first occurrence of silicon material for a 1d cutline taken at X=2.

```
extract name="max_y" max.bound material="silicon" x.val=2 mat.occno=1
```

Extracting the minimum X boundary (left side) location of the second occurrence of polysilicon material for a 1d cutline at Y=3.

```
extract name="min_x" min.bound material="polysilicon" y.val=3 mat.occno=2
```

2D Material Region Boundary

Extracting the minimum X boundary (left side) location of the photoresist material region at XY coordinates (7.6, -1.2).

```
extract name="minx" min.bound x.pos material="photoresist" x.val=7.6
y.val=-1.2
```

Extracting the maximum Y boundary (upper side) location of the photoresist material region at XY coordinates (5.2, 0).

```
extract name="maxy" max.bound y.pos material="photoresist" x.val=5.2
y.val=0
```

2D Concentration Area

Integrates the Boron concentration within the specified “box” limits, using a cutline step of 0.05 microns.

```
extract name="limit_area" 2d.area impurity="Boron" x.step=0.05 x.min=0.01
y.min=0.23 x.max=0.6 y.max=0.45
```

In addition to this statement, you can add the `interpolate` flag. When present, this flag causes the extraction to perform interpolation at the edges of the specified bounding box for min/max concentration and position.

Integrates the Phosphorus concentration for the whole 2D structure using a cutline step of 0.03 microns.

```
extract name="device_area" 2d.area impurity="Phosphorus" x.step=0.03
```

Note: The `x.step` refers to the number of 1d cutlines used to obtain the 2D area. For a device with an X axis of 7 microns, an `x.step` of 1 would result in 8 cutlines being used at 1 micron intervals.

2D Maximum Concentration File

Creates a Data format file plotting the position of the maximum potential, in silicon material only, for the whole 2D structure. A maximum potential Y position is found for every X step

of 0.1 microns. These Data format files can be loaded into TonyPlot (-ccd) to represent a line of maximum concentration across a device.

```
extract name="Total_max_pot" max.conc.file impurity="potential" x.step=0.1
material="silicon" outfile="totalconc.dat"
```

Creates a Data format file plotting the position of the maximum potential, in any material, for the specified “box” limits. A maximum potential Y position is found for every X step of 0.2 microns.

```
extract name="limit_max_pot" max.conc.file impurity="potential"
x.step=0.2 outfile="limitconc.dat" x.min=0 x.max=7 y.min=0 y.max=0.09
```

Note: The x.step does not refer to cutlines but to the number of X coordinates used. A value of 1 representing stepping 1 micron in X for every max Y value calculated.

QUICKMOS CV Curve

Extract a MOS CV curve, ramping the gate from 0 to 5 volts, with 0 volts on the backside and the device temperature set at 325 Kelvin (default 300 K). This example creates a curve that is stored in file cv.dat and can be shown using TonyPlot. To bring up TonyPlot on this file, an easy way is to highlight the file name and then click on DeckBuild’s **Tools** button. TonyPlot starts and loads with the file automatically.

```
extract name="CV curve" curve(bias,1dcapacitance vg=0.0 vb=0.0
bias.ramp=vg bias.step=0.25 bias.stop=5.0 x.val=0.1 temp.val=325) out-
file="cv.dat"
```

To get the maximum capacitance for the same curve, insert the keyword max (by editing the syntax created by the popup). Notice that in this example, a single value is being extracted from a curve, not the curve itself. You still, however, store the curve used during the calculation into an output file, which is always the case.

```
extract name="CV curve Max cap" max(curve(bias,1dcapacitance vg=0.0 vb=0.0
bias.ramp=vg bias.step=0.25 bias.stop=5.0 x.val=0.1 temp.val=325))
outfile="cv.dat"
```

To find what the capacitance was at voltage 4.3 volts, use the following syntax:

```
extract name="MOS capacitance at Vg=4.3" y.val from curve(bias,1dcapaci-
tance vg=0.0 vb=0.0 bias.ramp=vg bias.step=0.25 bias.stop=5.0 x.val=0.1
temp.val=325) where x.val = 4.3
```

The general form of this syntax is

```
extract y.val from curve(xaxis, yaxis) where x.val=number_on_xaxis
```

and:

```
extract x.val from curve(xaxis, yaxis) where y.val=number_on_yaxis
```

where xaxis and yaxis will determine the actual curve. The syntax for this example was created by using the popup to write the syntax for the CV curve, and then adding the y.val... where x.val syntax in the input deck.

For more examples on how to manipulate curves, see the examples in Section 5.4 “Device Extraction”.

Junction Capacitance Curve

Extract a curve of junction capacitance against bias where the first region in the top (first) layer of silicon is ramped from 0 to 5V. Capacitance of the first junction occurrence (upper) is measured and the resultant curve is output to the file XjV.dat. Device temperature is default (300 Kelvin). If only one junction exists for the selected region, you must use then a junction occurrence of one (upper).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=5.0 x.val=0.1 region.occno=1

extract done name="Junc cap vs bias" curve(bias,1djunc.cap material="Silicon" mat.occno=1 x.val=0.1 region.occno=1 junc.occno=1) outfile="XjV.dat"
```

Extract the minimum junction capacitance on the created junction capacitance against bias curve. The second region in the top (first) layer of silicon is ramped from 0 to 3V and the capacitance of the second junction occurrence (lower) is measured. Device temperature is set for calculations to be 325 Kelvin. The resultant curve is output to the file XjVmin.dat, while the extracted minimum value is logged to the default results file (results.final).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=3.0 x.val=0.1 region.occno=2

extract done name="Junc cap vs bias" min(curve(bias,1djunc.cap material="Silicon" mat.occno=1 x.val=0.1 region.occno=2 junc.occno=2 temp.val=325)) outfile="XjVmin.dat"
```

Note: The junction occurrence is only valid for the specified region. In other words, there is only a maximum of two possible junctions for the specified region.

Junction Breakdown Curve

Extract a curve of electron ionization integral against bias where the first region in the top (first) layer of silicon is ramped from 0 to 5V and device temperature is set to be 325 Kelvin. The resultant breakdown curve is output to the file Nbreakdown.dat. See the Impact command section and Impact Ionization physics section in the Atlas User's Manual for the Selberherr model used in calculation.

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.25
bias.stop=5.0 x.val=0.1 region.occno=1

extract done name="N Breakdown "curve(bias,n.ion material="Silicon" mat.occno=1 x.val=0.1 region.occno=1 temp.val=325)

outfile="Nbreakdown.dat"
```

The following extraction creates a curve of hole ionization integral against bias, and calculates the breakdown voltage corresponding to the point where the hole ionization integral intercepts 1.0. The second region in the top (first) layer of silicon is ramped from 0 to 20V and the device temperature is set to the default of 300 Kelvin. The resultant breakdown curve is output to the file Pbreakdown.dat and the breakdown voltage is appended to the default results file (results.final).

```
extract start material="Silicon" mat.occno=1 bias=0.0 bias.step=0.50
bias.stop=20.0 x.val=0.1 region.occno=2

extract done name="P intercept" x.val from curve(bias,p.ion material="Silicon" mat.occno=1 x.val=0.1 region.occno=2) where y.val=1.0

outfile="Pbreakdown.dat"
```

You can modify the selberherr model parameters using the syntax below. For more information, see Appendix Appendix A “Models and Algorithms,”.

```
extract start material="Silicon" mat.occno=1 bias=0.2 bias.step=0.08
bias.stop=5.0 x.val=0.3 region.occ=2
```

```
extract done name="iiP" curve(bias, p.ion material="Silicon" mat.occno=1
x.val=0.3 region.occno=2 egran=4.0e5 betap=1.0 betan=1.0
```

```
an1=7.03e5 an2=7.03e5 bn1=1.231e6 bn2=1.231e6 ap1=6.71e5 ap2=1.582e6
bp1=1.693e6 bp2=1.693e6) outfile="extract.dat"
```

SIMS Curve

Extract the concentration profile of net doping in the top (first) layer of silicon. The output curve is placed into the file SIMS.dat.

```
extract name="SIMS" curve(depth, impurity="Net Doping" material="Silicon"
mat.occno=1 x.val=0.1) outfile="SIMS.dat"
```

SRP Curve

Extract the **SRP** (Spreading Resistance Profile) in the top (first) silicon layer. The output curve is placed into the file SRP.dat.

```
extract name="SRP" curve(depth, srp materials="Silicon" mat.occno=1
x.val=0.1)
```

```
outfile="SRP.dat"
```

The following command will calculate the **SRP** (Spreading Resistance Profile) in the top (first) silicon layer using a specified 100 etch steps of uniform size. The output curve is placed into the file SRP100.dat.

```
extract name="SRP100" curve(depth, srp material="Silicon"
mat.occno=1 n.step=100 x.val=0.5) outfile="srp100.dat"
```

Note: Where n.step is not specified, the default is 50 etch steps of variable size dependent on the gradient of net concentration. If n.steps is set, uniform etch steps are used.

Sheet Resistance/Conductance Bias Curves

Extract the **Total** sheet conductance against bias curve of the first p-n region in the top (first) occurrence of polysilicon. Polysilicon is treated as a metal by default but is flagged here as a semiconductor (semi.poly). The device temperature is set to 325 Kelvin (default=300 Kelvin) and a bias ramped from 0 to 5V on the same polysilicon region.

```
extract start material="Polysilicon" mat.occno=1 bias=0.0 bias.step=0.00
bias.stop=5.0 x.val=0.1 region.occno=1
```

```
extract done name="Total SC" curve(bias, 1dconduct material="Polysilicon"
mat.occno=1 temp.val=325 x.val=0.1 region.occno=1 semi.poly) out-
file="totalSC.dat"
```

Extract the n-type sheet conductance against bias curve of the first p/n region in the top (first) occurrence of silicon where a bias ramped from 0V to 5V on the same silicon region and a value of QSS (4.0e10) is specified for the first interface occurrence.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont interface.occno=1 qss=4.0e10
extract done name="N-type SC" curve(bias,1dn.conduct material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1) outfile="NtypeSC.dat"
```

Extract the p-type sheet conductance against bias curve of the first p-n region in the top (first) occurrence of silicon, where a bias ramped from 0 to 5V on the same silicon region and a bias of 2V is held on the first region of the top occurrence of polysilicon. A value of QSS (5.0e10) is also specified for the first interface occurrence.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont material="Polysilicon" mat.occno=1 bias=2.0 x.val=0.1
region.occno=1
extract cont interface.occno=1 qss=5.0e10
extract done name="P-type SC" curve(bias,1dp.conduct material="Silicon"
mat.occno=1 x.val=0.1 region.occno=1) outfile="PtypeSC.dat"
```

The command below extracts the p-type sheet conductance against bias curve of the first and second p-n regions in the top (first) layer of silicon, where a bias is ramped from 1V to -2V on the top (first) polysilicon layer.

```
extract start material="Polysilicon" mat.occno=1 bias=1.0 bias.step=-0.05
bias.stop=-2.0 x.val=0.01
extract done name="region1+2" curve(bias,1dp.conduct material="Silicon"
mat.occno=1 x.val=0.01 region.occno=1 region.stop=2) out-
file="region1+2.dat"
```

Note: For sheet resistance extraction, substitute "1dconduct" with "1dsheet.res" (i.e., 1dsheet.res, 1dnsheet.res, 1dpsheet.res).

Electrical Concentration Curve

Extract the electron distribution against depth for the top (first) layer of silicon where a bias is ramped from 0 to 5V for the first region of the silicon and a QSS of 4.0e10 set for the first interface occurrence. Device temperature is set at 325 Kelvin.

```
extract start material="Silicon" mat.occno=1 region.occno=1 bias=0.0
bias.step=0.00 bias.stop=5.0 x.val=0.1
extract cont interface.occno=1 qss=4.0e10
extract done name="Electrical conc" curve(depth,n.conc material="Silicon"
mat.occno=1 x.val=0.1 temp.val=325) outfile="extract.dat"
```

ED Tree (Optolith)

Create a Data format file plotting a single branch of an ED tree for deviation of 10% from the datum, the specified critical dimension (CD) value of 0.5. The x.step defines the defocus

step to be used. 0.08 representing 8% of the total X axis range for each calculation. For each value of defocus at the specified critical dimension deviation, the value of dose is interpolated. Therefore, the resulting curve is dose against defocus for a critical dimension of 0.5 plus 10%.

```
extract name="ed+10" edcurve(da.value."DEFOCUS", da.value."CDs",
da.value."DOSE",dev=10 datum=0.5 x.step=0.08) outf="ed10.dat"
```

Note: If no `x.step` is specified the actual curve defocus points are used.

Elapsed time

The timer is reset to 10 seconds, a timestamp extracted before and then after a simulation. The elapsed time is then calculated by subtraction.

```
extract name="reset_clock" clock.time start.time = 10
extract name="t1" clock.time
<simulation>
extract name="t2" clock.time
extract name="elapsed_time" $t2 - $t1
```

Note: This extraction does not measure CPU time

5.4 Device Extraction

Device extraction always deals with a “logfile” that contains I-V information produced by a device simulator (such as Atlas). Therefore, it deals almost exclusively in curves. The following section show how to construct a curve or extract values on a curve for all possible devices. For the special case of MOS devices, both Atlas has a popup with a number of pre-defined MOS tests. See [Section 5.6 “MOS Device Tests”](#) for more information.

Device extraction also deals with structure files, which contain information saved by a device simulator (e.g., Atlas). You can extract this information by using the process extraction syntax style shown below. The following extracts the total electric field for silicon in a 1D cutline, where $x = 0.5$ for the loaded device structure file.

```
extract name="test" 2d.max.conc impurity="E Field" material="Silicon"
x.val=0.5
```

There are some differences between the syntax used by Extract and the syntax used by the Atlas output command. [Section 5.10 “Using Extract with Atlas”](#) shows these differences.

Extract allows you to construct a curve using separate X and Y axes. For each axis, you can choose the voltage or current on any electrode, the capacitance or conductance between any two electrodes, or the transient time for AC simulations. You can either manipulate the axes individually, such as multiplication or division by a constant, or combine axes in algebraic functions.

Note: The curve manipulation discussed is equally applicable to all curves, whether the curve came from process or device simulation. The only type-specific syntax relates to the curve axes. For example, gate voltage can't be extracted from a process simulator. If you try, then a warning message will appear.

5.4.1 The Curve

The basic element is always the curve. Once the curve is constructed, it can be used as is, by saving it to a file for use by TonyPlot, or as an Optimizer target, or it can be used as the basis for further extraction. For details on the `extract curve` syntax, see [Section 5.3.1 “Extract Syntax”](#).

To construct a curve representing voltage on electrode "emitter1" (on the X axis) versus current on electrode "base2", write:

```
extract name="iv" curve(v."emitter1", i."base2")
```

The first variable specified inside the parentheses becomes the X axis of the curve. The second variable becomes the Y axis. The `v."name"` and `i."name"` syntax is used for any electrode name — just insert the proper name of the electrode. The electrode name be defined previously (such as in the device deck, or previous to that in an Athena input deck using the `electrode` statement, or interactively in DevEdit). Electrode names may contain spaces but must always have quotation marks.

Transient time is represented by the keyword `time`.

```
extract name="It curve" curve(time, i."anode")
```

For Device temperature curves, use:

```
extract name="VdT" curve(v."drain", temperature)
```


For extracting a frequency curve use:

```
extract name="Idf" curve(i."drain", frequency)
```

To extract a capacitance or conductance curve, use this syntax:

```
extract name="cv" curve(c."electrode1"electrode2", v."electrode3")
```

and

```
extract name="gv" curve(g."electrode1"electrode2", v."electrode3")
```

For other electrical parameters (see Section 5.3.1 “Extract Syntax” section for valid electrical parameters) use the following syntax:

```
extract name="IdT" curve(elect."parameter", v."drain")
```

An `extract name` is given in each example. Although optional, it is always a good idea to name `extract` statements so they can be identified later. Names are always necessary for entering an `extract` statement in DeckBuild’s Optimizer, and for recognition by the VWF.

It is also possible to shift or manipulate curve axes. Each axis is manipulated separately. The simplest form of axis manipulation is algebra with a constant.

```
extract name="big iv" curve(v."gate"/50, 10*i."drain")
```

You can multiply, divide, add, or subtract any constant expression to each axis.

Curve axis can also be combined algebraically, similar to TonyPlot’s function capability:

```
extract name="combine" curve(i."collector", i."collector"/i."base")
```

All electrode values (current, voltage, capacitance, conductance) can be combined in any form this way.

Another curve type is `deriv()` used to return the derivative ($dydx$). For example, statement below will create the curve of $dydx$ gate bias and drain current plotted against and X axis of gate bias.

```
extract name="dydx" deriv(v."gate", i."drain") \
  outfile="dydx.dat"
```

It is also possible to calculate $dydx$ to the n th derivative as below.

```
extract name="dydx2" deriv(v."gate", i."drain", 2) \
  outfile="dydx2.dat"
```

To find local maxima and minima on a curve, limit the section of the curve X axis. The following statement extracts the maximum drain current, where gate bias is between the limits of 0.5 volts and 2.5 volts.

```
extract name="limit" max(curve(v."gate", i."drain", x.min=0.5
  x.max=2.5)) outf="limit.dat"
```

In addition, there are several operators which apply to curve axes. They are as follows:

```
abs(axis)
log(axis)
log10(axis)
sqrt(axis)
```

```
atan(axis)
-axis
```

For instance:

```
extract curve(abs(i."drain"), abs(v."gate"))
```

The operators can be combined. For example, `log10(abs(axis))`. These operators also work on curve axes from process simulation.

5.4.2 Curve Manipulation

A number of curve manipulation primitives exist:

```
min(curve)
max(curve)
ave(curve)
minslope(curve)
maxslope(curve)
slope(line)
xintercept(line)
yintercept(line)
area from curve
area from curve where x.min=X1 and x.max=X2
x.val from curve where y.val=k
y.val from curve where x.val=k
x.val from curve where y.val=k and val.occno=n
y.val from curve where x.val=k and val.occno=n
grad from curve where y.val=k
grad from curve where x.val=k
```

For details on Extract curve manipulation syntax, see Section 5.3.1 “Extract Syntax”.

For instance, using the BJT curve example above, you could find the maximum of I_c/I_b vs I_c , or maximum beta, by writing:

```
extract name="max beta" max(curve(i."collector", i."collector"/i."base"))
```

`max()`, `min()`, and `ave()` all work on the Y axis of the curve.

The sloped lines and intercepts often work together. The primitives `minslope()` and `maxslope()` can be thought of as returning a line. Extracting a line by itself has no meaning, so three other operators take a line as input. The operators are `slope()`, which returns the slope of the line, and `xintercept()` and `yintercept()`, which return the value where the line intercepts the corresponding axis.

For instance, a V_t test for MOS devices looks at a curve of $V_g(x)$ versus $I_d(y)$ and finds the X intercept of the maximum slope. Such a test would look like:

```
extract name="vt" xintercept(maxslope(curve(abs(v."gate",
abs(i."drain"))))
```

Some Vt tests take off Vd/2 from the resulting value. You could write:

```
extract name="vt" xintercept (maxslope (curve (abs (v."gate",
  abs (i."drain")))) - ave (v."drain")/2
```

Note that the last example uses:

```
ave (v."drain")/2
```

The max(), min(), and ave() operators can be used on both curves,

```
extract name="Iave" ave (curve (v."gate", i."drain"))
```

and also on individual curve axes,

```
extract name="Iave" ave (i."drain")
```

or even on axis functions:

```
extract name="Icb max" max (i."collector"/i."base")
```

You can also find the Y value on a curve for a given X value and the other way round. For example, to find the collector current (Y) for base voltage 2.3 (X), use:

```
extract name="Ic [Vb=2.3]" y.val from curve (abs (v."base"), abs (i."collec-
  tor")) where x.val = 2.3
```

Extract uses linear interpolation if necessary. If more than one point on the curve matches the condition, Extract takes the first one, unless you use the following syntax to specify the occurrence of the condition. This example would find the second Y point on the curve matching an X value of 2.3.

```
extract name="Ic [Vb=2.3]" y.val from curve (abs (v."base",
  abs (i."collector"))
  where x.val = 2.3 and val.occno =2
```

The condition used for finding an intercept can be a value or an expression and therefore use the min(), max(), and ave() operators. The following command creates a transient time against drain-gate capacitance curve and calculates the intercepting time where the capacitance is at its minimum value.

```
extract name="t at Cdrain-gate [Min]" x.val from curve (time,
  c."drain"gate")
  where y.val=min (c."drain"gate")
```

In addition to finding intercept points on curves, you can also calculate the gradient at the intercept, specified by either a Y or X value as shown below.

```
extract name="slope_at_x" grad from curve (v."gate", i."drain")
  where x.val=1.5
extract name="slope_at_y" grad from curve (v."gate", i."drain")
  where y.val=0.001
```

You can also find the area under a specified curve for either the whole curve or as below between X limits.

```
extract name="iv area" area from curve (v."gate", c."drain"gate")
  where x.min=2 and x.max=5
```

5.4.3 BJT Example

As a final example for device extraction, consider finding, say, the beta value for a BJT device, at 1/10th the current for max beta. This example sums up the information presented so far, and also introduces the feature of variable substitution.

First, you need to figure out what the current is at max beta. Max beta was presented in a previous example:

```
extract name="max beta" max(curve(i."collector", i."collector"/i."base"))
```

After this statement has been run, `extract` remembers the extract name, `max beta`, and the resulting value. Use this information later on using variable substitution. In this example, you need to get the current, or X axis value, at `max beta`, to figure out what 1/10th of it is. To do this, use the extracted `max beta` as our Y axis “target value”:

```
extract name="Ic[max beta]" x.val from curve(i."collector",  
i."collector"/i."base") where y.val="$max beta"
```

Finally, extract the value of I_c/I_b for $I_c = \text{max beta}/10$.

```
extract name="Ic/Ib for Ic=Ic[max beta]/10" y.val from curve(i."collec-  
tor",  
i."collector"/i."base") where x.val="$Ic[max beta]"/10
```

For more information about variable substitution, see [Section 5.8 “Extract Features”](#).

5.5 General Curve Examples

The following examples assume that they are extracting values from the currently loaded logfile running under DeckBuild. Saved “IV” log files, however, can be used directly with `extract` using the syntax below.

```
extract init infile="filename"
```

Note: You can enter `extract` commands on multiple lines using a backslash character for continuation. You should, however, enter the syntax shown below on a single line although shown on two or more lines.

5.5.1 Curve Creation

The following command extracts a curve of collector current against base voltage and places the output in `icvb.dat`.

```
extract name="IcVb curve" curve(i."collector", v."base") out-
file="icvb.dat"
```

5.5.2 Min Operator with Curves

The following command calculates the minimum value for a curve of drain current against internal gate voltage.

```
extract name="Vgint [Min]" min(curve(i."drain", vint."gate"))
```

5.5.3 Max Operator with Curves

The following command calculates the maximum value for a curve of base voltage against base-collector capacitance.

```
extract name="Cbase-coll [Max]" max(curve(v."base", c."base"collector"))
```

5.5.4 Ave Operator with Curves

The following command calculates the average value for a curve of drain current against gate-drain conductance.

```
extract name="Ggate-drain [Ave]" ave(curve(i."drain", g."gate"drain"))
```

The preceding assumes that the x values (the drain current in this case) are equally spaced. If this is not so, the use of linear interpolation may be specified. Inserting the `interpolate` keyword thus :

```
extract name="Ggate-drain [Ave]" ave(curve(i."drain", g."gate"drain),
interpolate)
```

causes the number of interpolation points to be set to twice the number of points in the curve. Alternatively, the number of interpolation points may be specified explicitly, thus :

```
extract name="Ggate-drain [Ave]" ave(curve(i."drain", g."gate"drain),
interpolate n.step=100)
```

5.5.5 X Value Intercept for Specified Y

The following command creates a frequency against drain current curve and calculates the intercepting frequency for a drain current of 1.5×10^{-6} .

```
extract name="Freq at Id=1.5e-6" x.val from curve(frequency, i."drain")
where y.val=1.5e-6
```

5.5.6 Y Value Intercept for Specified X

The following command creates a drain voltage against device temperature curve and calculates the intercepting temperature for a drain voltage of 5V.

```
extract name="T at Vd=5" y.val from curve(v."drain", temperature) where
x.val=5.0
```

5.5.7 Abs Operator with Axis

The following command creates a curve of absolute gate voltage against absolute optical wavelength (log, log10 and sqrt also available).

```
extract name="Vg-optW curve" curve(abs(v."gate"), abs(elect."optical wave-
length"))
```

5.5.8 Min Operator with Axis Intercept

The following command creates a transient time against gate-drain capacitance curve and calculates the intercepting time where the capacitance is at its minimum value.

```
extract name="t at Cgate-drain[Min]" x.val from curve(time,
c."gate""drain") where y.val=min(c."gate""drain")
```

5.5.9 Max Operator with Axis Intercept

The following command creates a collector current against collector current divided by base current curve and calculates the intercepting collector current where I_c/I_b is at a maximum value.

```
extract name="Ic at Ic/Ib[Max]" x.val from curve(i."collector", i."collec-
tor"/i."base") where y.val=max(i."collector"/i."base")
```

5.5.10 Second Intercept Occurrence

The following command creates a gate voltage against source photo current curve and calculates the second intercept of gate voltage for a source photo current of $2e-4$.

```
extract name="2nd Vg at Isp=2e-4" x.val from curve(v."gate", elect."source
photo current") where y.val=2e-4 and val.occno=2
```

5.5.11 Gradient at Axis Intercept

The following command creates a probe Itime against drain current curve and finds the gradient at the point where probe Itime is at a maximum.

```
extract name="grad_at_maxTime" grad from curve(probe."Itime",
i."drain") where y.val=max(probe."Itime")
```

5.5.12 Axis Manipulation with Constants

The following command creates a gate voltage divided by ten against total gate capacitance multiplied by five. Adding and subtracting are also available.

```
extract name="Vg/10 5*C-gg curve" curve(v."gate"/10, 5*c."gate""gate")
```

5.5.13 X Axis Interception of Line Created by Maxslope Operator

The following command calculates the X axis intercept for the maximum slope of a drain current against gate voltage curve.

```
extract name="Xint for IdVg" xintercept(maxslope(curve(i."drain",
v."gate"))))
```

5.5.14 Y Axis Interception of Line Created by Minslope Operator

The following command calculates the Y axis intercept for the minimum slope of a substrate current against drain voltage.

```
extract name="Yint for IsVd" yintercept(minslope(curve(i."substrate",
v."drain")))
```

5.5.15 Axis Manipulation Combined with Max and Abs Operators

The following command calculates the maximum value of drain-gate resistance.

```
extract name="Rdrain-gate [Max]" max(1.0/(abs(g."drain" "gate")))
```

5.5.16 Axis Manipulation Combined with Y Value Intercept

The following command creates a gate voltage against drain-gate resistance and calculates the intercepting drain-gate resistance for a gate voltage of 0V.

```
extract name="Rdrain-gate at Vg=0" y.val from curve (v."gate", 1.0/
abs(g."drain" "gate"))
where x.val=0.0
```

5.5.17 Derivative

The following command creates the curve of dydx gate bias and drain current plotted against and X axis of gate bias.

```
extract name="dydx" deriv(v."gate", i."drain")
outfile="dydx.dat"
```

This further example calculates to the 2nd derivative.

```
extract name="dydx2" deriv(v."gate", i."drain", 2)
outfile="dydx2.dat"
```

5.5.18 Data Format File Extract with X Limits

The following command finds the local maximum in Data Format file for the curve of vin between 2 and 5 volts against power.

```
extract name="max [2-5]" max(curve(da.value."vin", da.value."power",
x.min=2 x.max=5)) outf="max2-5.dat"
```

5.5.19 Impurity Transform against Depth

The following command calculates the electron concentration in the first occurrence of silicon material for a cutline of X=1 squared against depth.

```
{fixed} extract name="nconc^2" curve(depth, (n.conc material="Silicon"  
mat.occno=1 x.val=1) * (n.conc material="Silicon" mat.occno=1 x.val=1))  
outfile="nconc.dat"
```


5.6 MOS Device Tests

A list of ready-made MOS extract statements is also provided. Use them directly or make modifications to suit testing needs. DeckBuild allows you to create, modify, and save tests.

The following MOS tests are:

- Vt
- Beta
- Theta
- Leakage
- Bvds
- Idsmax
- SubVt
- Isubmax
- Vg[Isubmax]

Do the following to access the list of MOS extract routines.

- **ATLAS:** Choose **Commands**→**Extracts**→**Device...** and the Atlas Extraction popup will appear. Choose the desired test and click on the **WRITE** button to insert the test into the input deck. Using the **User defined** option, you can enter custom extracts into the popup and save them as defaults.

When you click the **Write Deck** button on the **Control** popup, the `extract` syntax will be written automatically to the deck along with the selected tests (Figure 5-5).

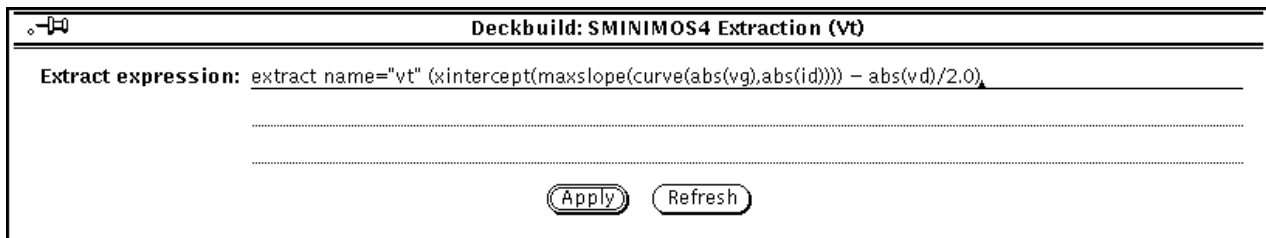


Figure 5-5 The Atlas Extraction (Vt) Popup

5.7 Extracted Results

Extracted results appear both with the simulator output in the **tty** subwindow and in a special file named by default `results.final`. You can name the file using the `datafile="filename"` syntax. Use the file to compare the results from a large number of runs. For example, if using DeckBuild's built-in optimizer, the file gives a concise listing of all the results as a function of the input parameters. The extract results file is created in the current working directory.

5.7.1 Units

- Material thickness (angstroms)
- Junction depth (microns)
- Impurity concentrations (impurity units, typically atoms/cm³)
- Junction capacitance (Farads/cm²)
- QUICMOS capacitance (Farads/cm²)
- QUICKMOS 1D V_t (Volts)
- QUICKBIP 1D solver (see the QUICKBIP section)
- Sheet resistance (Ohm/square)
- Sheet conductance (square/Ohm)
- Electrode voltage (Volts)
- Electrode internal voltage (Volts)
- Electrode current (Amps)
- Capacitance (Farads/micron)
- Conductance (1/Ohms)
- Transient time (Seconds)
- Frequency (Hertz)
- Temperature (Kelvin)
- Luminescent power (Watts/micron)
- Luminescent wavelength (Microns)
- Available photo current (Amps/micron)
- Source photo current (Amps/micron)
- Optical wavelength (Microns)
- Optical source frequency (Hertz)
- Current gain (dB)
- Unilateral power gain frequency (dB)
- Max transducer power gain (dB)

If desired, you can perform whatever unit shifting required by adding the appropriate constants in the device extract tests and saving them as the default. The units are always printed out along with the extract results for built-in single value extract routines. Custom extract routines do not show units.

5.8 Extract Features

5.8.1 Extract Name

`extract` statements should almost always be given names. The name must be prepended to the remainder of the `extract` statement. For example:

```
extract name="gateox thickness" oxide thickness x.val=1.0
```

The `extract` name is used in three ways. The name appears on the Optimizer worksheet when you enter the `extract` statement as a target, and on the VWF worksheet as an extracted parameter. It can also be used in further `extract` statements to perform variable substitution. The name can contain spaces.

5.8.2 Variable Substitution

The `extract` parser maintains a list of variables, each of which consists of a name and a value. A name is defined by any previous named `extract` statement. The corresponding value is the result of the statement.

To refer to a variable's value, precede it with a `'$'`. Quotes are optional around variable references, except when the variable name contains spaces, in which case the `$` must precede the quotes. The substituted variable acts as a floating point number, and can be used in any `extract` expression that uses numerical arguments.

For example:

```
extract name="xj1" xj silicon junc.occno=1
extract name="xj2" xj silicon junc.occno=2
extract name="deltaXj" abs($xj1 - $xj2)
```

Examples with spaces:

```
extract name="max boron" max.conc boron
extract name="max arsenic" max.conc arsenic
extract name="PN ratio" $"max boron"/$"max arsenic"
```

You can also use variable substitution in `extract` with the `set` command as shown below.

```
set cutline=0.5
extract name="gateox thickness" oxide thickness x.val=$cutline
```

In addition, filenames to be loaded can also be specified this way. For example:

```
set efile = structure.str
extract init infile="$'efile'"
```

Note: Single quotes can be used to substitute where `$`-variable must appear within double quotes.

5.8.3 Min and Max Cutoff Values

Statements may contain `min.val=value` or `max.val=value` or both to define a valid range for extracted results (single-valued results only, not curves). If you do not define either `max` or `min`, then the range extends from $+\infty$ to the stated value respectively. If the extracted value is outside the range, then an error message is printed along with the extracted results and also appended to the default results file.

5.8.4 Multi-Line Extract Statements

Extract statements may be spread over multiple lines to specify layer biases and QSS values as shown in above examples. This involves using the `start/cont/done` syntax.

5.8.5 Extraction and the Database (VWF)

When run with the Virtual Wafer Fab, all extract values in the deck appear as output result columns on the split worksheet. Each row of the worksheet contains the input parameters used to create the results. The extracted value cell values are filled in automatically as the split points complete. If some extracts are only intermediate calculations and are not required to be included in the results worksheet the `hide` flag can be used. This prevents unrequired extract results from cluttering the worksheet data.

The `min/max extract` ranges, if defined, are examined. If any extracted value is out of range, then children of that deck fragment (any part of the worksheet that uses the simulation results of that deck fragment) are automatically de-queued and marked with a parent error. The fragment is marked with a range error. The purpose here is that the system does not waste its time by running any simulation beyond that point in the input deck where the range error occurred, for all parts of the split tree that use the particular values of the deck.

5.9 QUICKBIP Bipolar Extract

QUICKBIP is a 1D simulator for bipolar junction transistors (BJT) and is fully integrated inside the DeckBuild environment. It is accessed using the `extract` command and is available for use with any Silvaco simulator.

The doping profile passed to the QUICKBIP solver should be a bipolar profile. At least, three regions must exist. The top region in the first silicon layer is taken to be the emitter. There may be other materials on top of the silicon.

QUICKBIP can be used with either Athena (2D process simulation) or SSuprem3 (1D process simulation). It is used in cases where a 1D device simulation is both easier and faster to turn around a result. Examples of the QUICKBIP `extract` command language are listed as follows:

```
extract name="bip test bf" bf
extract name="bip test nf" nf
extract name="bip test is" gpis
extract name="bip test ne" ne
extract name="bip test ise" ise
extract name="bip test cje" cje
extract name="bip test vje" vje
extract name="bip test mje" mje
extract name="bip test rb" rb
extract name="bip test rbm" rbm
extract name="bip test irb" irb
extract name="bip test tf" tf
extract name="bip test cjc" cjc
extract name="bip test vjc" vjc
extract name="bip test mjc" mjc
extract name="bip test ikf" ikf
extract name="bip test ikr" ikr
extract name="bip test nr" nr
extract name="bip test br" br
extract name="bip test isc" isc
extract name="bip test nc" nc
extract name="bip test tr" tr
```

Any name can be assigned to each command. In the case of a 2D simulator, the lateral position of the vertical profile has to be specified with the parameter `x.val=n`. For example:

```
extract name = "forward transit time" tf x.val=0.3
```

Alternatively, a boolean region can be specified when running in conjunction with the IC Layout interface. For example:

```
extract name="my test" tf region="pnp_active_poly"
```

In this case, the bipolar test is performed only in the case where an IC layout cross section intersects the named region.

You can modify QUICKBIP tuning parameters for using the syntax shown below. [Appendix A “Models and Algorithms”](#) provides a more detailed explanation.

```
extract name="Tuning bf" bf x.val=0.5 bip.tn0=1.0e-5 bip.tp0=1.0e-3
bip.an0=2.9e-31 bip.ap0=0.98e-31 bip.nsrhn=5.0e12 bip.nsrhp=5.0e15
bip.betan=2.1 bip.betap=1.
```

[Table 5-1](#) shows the extract parameters representing the BJT parameters.

Table 5-1 BJT Parameters		
Parameter	Description	Units
bf	Ideal Maximum Forward Beta	
nf	Forward current Emission Coefficient	
gpis	Transport saturation current (IS)	A/cm ²
ne	Base-Emitter Leakage Emission Coefficient	
ise	Base-Emitter Leakage Saturation Current	A/cm ²
cje	Base-Emitter Zero Bias DEpletion Capacitance	F/cm ²
vje	Base-Emitter built in potential	V
mje	Base-Emitter exponential factor	
rb	Zero bias base resistance	Ohms/square
rbm	Minimum base resistance at high current	Ohms/square
irb	Current at half base resistance value	A/cm ²
tf	Ideal forward transit time (1/ft)	secs
cjc	Base-Collect zero bias depletion capacitance	F/cm ²
vjc	Base-Collector built in potential	V
mjc	Base-Collector exponential factor	
ikf	Corner of Forward Beta High current roll-off	A/cm ²
ikr	Corner of Reverse Beta High current roll-off	A/cm ²
nr	Reverse Current Emission Coefficient	
br	Ideal Maximum Reverse Beta	

Table 5-1 BJT Parameters

Parameter	Description	Units
isc	Base-Collector Leakage Saturation Current	A/cm ²
nc	Base-Emitter Leakage Emission Coefficient	
tr	Ideal forward transit time	secs

Automated command writing is accomplished with the use of the **DeckBuild Extract** popup window. This is accessed from the **Commands** menu when either SSuprem3 or Athena is selected as the current simulator.

I-V Curves can be visualized with TonyPlot if the **Compute I-V curve** option is selected on the Extract popup. In this case, select from either forward or reverse characteristics and specify the axes of the curve.

- All extracted parameters can be used as optimization targets.
- All extracted parameters are appended to the default results file in the current working directory. Unless specified, using the `datafile=filename` syntax, it defaults to `results.final`.
- When running under the VWF, all extracted parameters will be logged for regression modeling.

QUICKBIP solves fundamental system of semiconductor equations, continuity equations for electrons and holes, and Poisson's equation for potential self-consistently using the Gummel method. The following physical models are taken into account by QUICKBIP:

- Doping-dependent mobility
- Electric field dependent mobility
- Band gap narrowing
- Shockley-Read-Hall recombination
- Auger recombination

QUICKBIP is fully automatic so that it is unnecessary to specify input biases. QUICKBIP calculates both forward and inverse characteristics of the BJT. For an n-p-n device, these sets are as follows:

1. $V_{eb} = -0.3 \dots -V_{eb_final}$, $V_{eb_step} = -0.025$, $V_{cb} = 0$ V
2. $V_{cb} = -0.3 \dots -V_{cb_final}$, $V_{cb_step} = -0.025$, $V_{eb} = 0$ V
3. V_{eb_final} and V_{cb_final} depend on the particular BJT structures, usually about $-1 \dots -1.5$ (high injection level).

For a p-n-p device, all signs are changed.

5.10 Using Extract with Atlas

Do the following to calculate `extract` parameters during an Atlas simulation.

1. Include an output statement in your original input deck that specifies the parameters of interest (e.g., output charge to specify charge concentration). You cannot extract a parameter unless you specify that parameter (either explicitly or by default) in an output statement.
2. Insert an `extract` statement to extract the desired parameters (see [Chapter 5 “Extract”](#)).

There are some differences between the Extract syntax and the syntax used by the Atlas output statement. To extract parameters, use the correct `extract` statement syntax (not the Atlas Output statement syntax). For example, the Atlas `OUTPUT` statement uses `E.Field` to specify electric field, while the `extract` statement requires the name `E.Field`. To extract electric field include the following lines in the input deck:

```
Output E.Field
...
...
...
Extract ... Impurity="E Field"
```

The following table shows the differences between the Atlas syntax and the `extract` statement syntax.

Atlas Parameter	Atlas Default	Extract Parameter	Units
POTENTIAL	Mandatory	Potential	V
NET DOPING	Mandatory	Net Doping	atoms/cm ³
ELECTRON CONCENTRATION	Mandatory	Electron Conc	cm ³
HOLE CONCENTRATION	Mandatory	Hole Conc	cm ³
CHARGE	False	Change Conc	atoms/cm ³
CON.BAND	False	Conduction Band Energy	V
E.FIELD/EFIELD	False	E Field	V/cm
E.MOBILITY	False	e- Mobility	cm ² /Vs
E.TEMP	True	Electron Temp	K
E.VELOCITY	False	Electron Velocity	cm/s
EX.FIELD	True	E Field X	V/cm
EX.VELOCITY	False	e- Velocity X	m/s
EY.FIELD	False	E Field Y	V/cm
EY.VELOCITY	False	e- Velocity Y	m/s
FLOWLINES	False	Current Flow	None
H.MOBILITY	False	h+ Mobility	cm ² /Vs

Atlas Parameter	Atlas Default	Extract Parameter	Units
H . TEMP	True	Hole Temp	K
H . VELOCITY	False	Hole Velocity	cm/s
HX . VELOCITY	False	h+ Velocity X	m/s
HY . VELOCITY	False	h+ Velocity Y	m/s
IMPACT	True	Impact Gen Rate	scm ³
JY . ELECTRON	False	Je- Y	A/cm ²
J . ELECTRON	True	Je- Current Magnitude	A/cm ²
JX . ELECTRON	FALSE	Je- X	A/cm ²
J . CONDUCT	True	Conduction Current	A/cm ²
J . DISP	False	Displacement Current	A/cm ²
J . HOLE	True	h+ Current Magnitude	A/cm ²
J . TOTAL	True	Total Current Density	A/cm ²
JX . CONDUCT	False	Cond Current X	A/cm ²
JX . HOLE	False	Jh+ X	A/cm ²
JX . TOTAL	False	Jtot X	A/cm ²
JY . CONDUCT	False	Cond Current Y	A/cm ²
JY . HOLE	False	Jh+ Y	A/cm ²
JY . TOTAL	False	Jtot Y	A/cm ²
PHOTOGEN	True	Photo Generation Rate	scm ³
QFN	True	Electron QFL	V
QFP	True	Hole QFL	V
QSS	False	Interface Charge	cm ²
RECOMB	True	Recombination Rate	scm ³
TOT . DOPING	False	Total Doping	atoms/cm ³
TRAPS	True	Traps	cm ³
U . AUGER	False	Auger Recomb Rate	scm ³
R . RADIATIVE	False	Radiative Recomb Rate	scm ³
U . SRH	False	SRH Recomb Rate	scm ³
VAL . BAND	False	Valence Band Energy	V
X . COMB	False	Composition X	None
Y . COMB	True	Composition Y	None

Atlas Parameter	Atlas Default	Extract Parameter	Units
OPT . INTENS	False	Optical Intensity	W/cm ²
OX . CHARGE	False	Fixed Oxide Charge	cm ³



Chapter 6 Optimizer

6.1 Overview

An optimizer is a mechanism that automatically varies one or more input parameters and carries out simulations in order to match one or more targets. The Optimizer thereby runs through a number of iterations until the results match the targets within a certain tolerance.

The Optimizer in Deckbuild uses the Levenberg-Marquart algorithm from the MINPACK optimization library to build a response surface of results versus input parameters as the iterations progress. This response surface is used to calculate the input parameter values for each iteration. If – for some reason – the Optimizer cannot achieve convergence, it stops and displays the error condition.

In practice, several parameters are measured and the simulation is then tuned to those values. For example, MOS structure, gate oxide thickness, can all be used to tune the input deck with the Optimizer.

6.1.1 Features

The Optimizer eliminates guesswork by determining the input parameter values necessary to match one or more targets quickly and accurately. Furthermore, since the Optimizer is built on top of DeckBuild's native auto-interfacing capability, parameters in one simulator can be optimized against the extracted results from a different simulator.

The Optimizer in Deckbuild is most useful for tuning studies. Use it to calibrate extracted simulation results to measured data by changing coefficients, such as diffusion and segregation; rather than trying to vary settings, such as simulated time and temperature. You can use such well-tuned input deck fragments, each of which performs a known operation, to build entire input decks.

For design studies, rather than tuning studies, we recommend the Virtual Wafer Fab (VWF). The VWF constructs and allows visualization of the entire process response surface, opposed to meeting a single target. It also acts in unison with the Optimizer by storing the well-tuned input deck operations for later use. Since the input deck requires no modification and no special statements, you can easily optimize any existing deck. When there are satisfying optimization results, the Optimizer can copy the final parameter values back into the deck. You can save all parameter and target data from the worksheet to disk and reload it at any time.

For the Deckbuild Optimizer any variable defined via a set statement can serve as an optimization parameter and any extracted value can be used as a target. You can optimizer several targets at the same time and also use curves as targets.

6.1.2 Terminology

This chapter makes reference to input parameters and targets. Input parameters, or just parameters, are any numerical constants in the input deck. Examples include implant energy, diffusion time, and gate voltage. Targets are values that are extracted from the simulated results. Examples include oxide thickness or V_t .

6.2 Using the Optimizer

In the following we will demonstrate how an optimization can be run in Deckbuild. We will explain the features of the optimizer by means of an example. The threshold voltage V_t of a MOS transistor shall be optimized towards a selected value by varying parameters of the threshold adjust implant. This optimization example is based on the mos1ex01 standards example that is shipped with Deckbuild. The first step will be to define the parameters that we want to optimize. To do so, open the deck and introduce two variables by using the set statement as shown in this code snippet.

```
#
set vt_dose=9.5e11
set vt_energy=10

#vt adjust implant
implant boron dose=$vt_dose energy=$vt_energy pearson
```

The next step is then to write the optimizer script file. The syntax of the file is XML.

```
<optimization>
  <parameter-list>
    <parameter name="vt_dose" nom="9e11" min="4.5e11" max="1.9e12"></parameter>
    <parameter name="vt_energy" nom="10.0" min="5.0" max="15.0"></parameter>
  </parameter-list>
  <target-list>
    <target name="nvt" value="0.65"/>
  </target-list>
</optimization>
```

The example script file shows two major sections, one is the `<parameter-list>`, the other is the `<target-list>`. In the `<parameter-list>` section we have to define the parameters we would like the optimizer to vary and also the minimum, and maximum values as well as the starting (nominal) value. The `<target-list>` defines a list of targets to optimize to including their values. In this case a single target called `nvt` with value of 0.65V was defined. The names of parameters and targets must be equal to the names used in set and extract statements.

After the file was created you can then run the optimization by using a command as follows:

```
deckbuild -run -ascii ./mos1ex01.in -opt ./opt.xml
```

This will start deckbuild in optimizer mode. You will see the usual (for batch mode) runtime output appear on the console window. You can stop the optimization/simulation anytime by hitting Ctrl-C. Additionally to the runtime output, two files called `progress.opt` and `results.opt` are created. The former is updated with every iteration and contains information about the progress of the optimization run. You can open a 2nd terminal window and watch this file as it changes. It contains entries of the form:

```
OPTIMIZER SETTINGS
ftol:0.1
gtol:0
xtol:0.1
maxfev:800
epsfcn:0.001
factor:100
TARGETS
target-name:nvt,simulated-file:
reference-values:0.65
```

```
target-name:dummy,simulated-file:
reference-values:0
EVALUATION:1
PARAMETERS
vt_dose:9.12471e+11,vt_energy:10
nvt:0.523108,residual:-0.126892
```

When the optimization has reached a truncation criterion deckbuild will stop. The file results .opt will then contain the results obtained from the optimizer:

```
vt_dose=1.2162e+12
vt_energy=8.29531
RETURN:2
TOTAL EVALUATIONS:11
```

In this case the values for the two parameters vt_dose and vt_energy is shown as well as the return status of the optimizer and the number of iterations that were needed.

6.2.1 Parameter settings

The <parameter> element supports the following settings (attributes)

- name="vt_dose". Defines the name of the parameter as it appears in the set statement
- nom="9e11". The nominal (or initial) value of the parameter. The optimizer runs the first simulation using the nominal value
- min="4.5e11". The minimum value of the parameter.
- max="1.9e12". The maximum value of the parameter
- scale="log". Scaling of the parameter. This can be set to either "log" (logarithmic) or "lin" (linear). Omitting the parameter implies a setting of "lin". A linear scaling means that the parameter is passed to the optimizer as is, whereas a logarithmic scaling means the log value of the parameter is passed to the optimizer.

6.2.2 Target settings

The <target> element supports the following settings (attributes)

- name="vt". The name of the target as it appears in the extract statement
- type="curve". The type of the target. Can either be "scalar" or "curve".
- reference="log_ref.dat". Valid in case the target is of type "curve". Defines a curve, which denotes the measurement. This is taken as the reference curve for the optimizer
- simulated="out.dat". Valid in case the target is of type "curve". Defines a curve, which denotes a simulation result. The deck, which is run by the optimizer must contain simulator statements to create this file.
- scale="log". Scaling of the target. Can be set to either "lin" (linear) or "log" (logarithmic). The setting influences how the error vector is computed. A linear scaling means the absolute error of the computed and measured target value(s) is computed. A logarithmic setting means that the absolute error of the logarithm of the computed and measured target value(s) is computed.
- weights="weighs.dat". An optional DAT file containing a weight. The weight allows to emphasize certain parts of a curve over other parts of the same curve. The DAT file must have as many entries as are in the curve file. weighing is applied after the error vector was computed. The error vector is multiplied by the given weight. If this parameter is not given a default value of 1 is implied.

Below an example weighs file containing 10 values is given:

```
weigh
10 2 2
index
value
0 1.0000000000000000e+00
1 2.0000000000000000e+00
2 3.0000000000000000e+00
3 4.0000000000000000e+00
4 5.0000000000000000e+00
5 4.0000000000000000e+00
6 3.0000000000000000e+00
7 2.0000000000000000e+00
8 1.0000000000000000e+00
9 0.0000000000000000e+00
```

6.2.3 Settings of the Optimizer

The optimizer has various settings that can be used to influence the optimization run. They are defined via the XML file and have to appear in a separate section called `<settings>`. If a parameter is not given in the file a default value is used. As shown in above example the `<settings>` section can be omitted as a whole in which case default values will be used for all parameters.

All settings are passed on to the underlying MINPACK algorithm.

The following settings are supported:

```
<settings>
  <setting name="epsfcn" value="0.001" />
  <setting name="xtol" value="0.1" />
  <setting name="gtol" value="0.0" />
  <setting name="ftol" value="0.1" />
  <setting name="maxfev" value="800" />
</settings>
```

Table 6-1 Optimizer Settings

Name	Description	Default
epsfcn	epsfcn is an input variable used in determining a suitable step length for the forward-difference approximation. This approximation assumes that the relative errors in the functions are of the order of epsfcn. If epsfcn is less than the machine precision, it is assumed that the relative errors in the functions are of the order of the machine precision.	0.001
xtol	xtol is a nonnegative input variable. Termination occurs when the relative error between two consecutive iterates is at most xtol. Therefore, xtol measures the relative error desired in the approximate solution.	0.1
gtol	gtol is a nonnegative input variable. Termination occurs when the cosine of the angle between fvec and any column of the jacobian is at most gtol in absolute value. Therefore, gtol measures the orthogonality desired between the function vector and the columns of the jacobian.	0.0
ftol	ftol is a nonnegative input variable. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares.	0.1
maxfev	maxfev is a positive integer input variable. Termination occurs when the number of evaluations (simulations) is at least maxfev by the end of an iteration.	800

6.2.4 Running optimizations on curves

The optimizer supports optimizing on curve targets. This works by setting the type of a parameter to being a curve as in this code snippet:

```
<target-list>
  <target name="cc" type="curve" reference="log_ref.dat" simulated="out.dat" />
</target-list>
```

Above target definition defines a target named CC to be of type curve. The target values are specified using the “reference” attribute (`log_ref.dat` in this case). The simulator created (extracted) curve is defined using the “simulated” attribute (`out.dat` in this case). The optimizer will compute an error vector of the two given vectors using a component-wise relative error. Note, that the dimension of the two vectors must be equal. You need to use a script in your simulation deck, which takes care of interpolating missing values if needed.

6.2.5 Optimizer return values

When an optimization finishes, the optimizer returns with a certain status. The status is indicated via the RETURN value written into the results.opt file. The following table lists all possible return values and their meaning:

Value	Description
0	Improper input parameters.
1	Both actual and predicted relative reductions in the sum of squares are at most ftol.
2	Relative error between two consecutive iterates is at most xtol.
3	Conditions for info = 1 and info = 2 both hold.
4	The cosine of the angle between the error vector and any column of the jacobian is at most gtol in absolute value.
5	Number of calls to fcn has reached or exceeded maxfev.
6	ftol is too small. No further reduction in the sum of squares is possible.
7	xtol is too small. No further improvement in the approximate solution x is possible.
8	gtol is too small. fvec is orthogonal to the columns of the jacobian to machine precision.



Appendix A

Models and Algorithms

A.1 Introduction

Models and Algorithms used by one dimensional (1D) electrical solvers in DeckBuild and TonyPlot.

Note: This appendix is intended to serve as a quick reference only. A detailed description of the semiconductor device physical models is provided in the Atlas manual.

1D electrical solvers, available by using the `extract` command in DeckBuild or in TonyPlot, are based on the iterative solution of the Poisson equation:

$$\text{div}(\varepsilon \nabla \psi) = q(p - n + N_D^+ - N_A^-) - \rho_F \quad \text{A-1}$$

where ψ is the potential, ε is the dielectrical permittivity, n and p are the electron and hole concentrations, and ρ_F is the fixed charge.

QUICKBIP uses the continuity equations to calculate n and p :

$$\frac{1}{q} \text{div} J_n^\otimes - U_n = 0 \quad \text{A-2}$$

$$\frac{1}{q} \text{div} J_p^\otimes - U_p = 0 \quad \text{A-3}$$

where:

$$J_n^\otimes = q\mu_n E_n^\otimes \cdot n + qD_n \nabla n \quad \text{A-4}$$

$$J_p^\otimes = q\mu_p E_p^\otimes \cdot p + qD_p \nabla p \quad \text{A-5}$$

$$D_n = \frac{kT}{q} \mu_n, \quad D_p = \frac{kT}{q} \mu_p \quad \text{A-6}$$

A.1.1 Physical Models

All electrical solvers take into account the following models and effects:

- Temperature dependence, such as kT/q or E_g
- Concentration-dependent mobility (with built-in temperature dependence)
- Field-dependent mobility (perpendicular field with built-in temperature dependence)
- Material work function (for MOS structures)
- Fixed interface charge

A.2 Concentration Dependent Mobility

The concentration dependent mobilities for n and p respectively are:

$$\mu_n^D = \mu_{nmin} + \frac{\Delta\mu_n}{1 + N_{total}/N_{nref}} \quad \text{A-7}$$

$$\mu_p^D = \mu_{pmin} + \frac{\Delta\mu_p}{\Delta + N_{total}/N_{pref}} \quad \text{A-8}$$

where:

$$\mu_{nmin} = 88 \cdot \left(\frac{Y}{300}\right)^{-0.57} \quad \text{A-9}$$

$$\mu_{pmin} = 54.3 \cdot \left(\frac{Y}{300}\right)^{-0.57} \quad \text{A-10}$$

$$\Delta\mu_n = 1252 \cdot \left(\frac{Y}{300}\right)^{-2.33} \quad \text{A-11}$$

$$\Delta\mu_p = 407 \cdot \left(\frac{Y}{300}\right)^{-2.33} \quad \text{A-12}$$

$$N_{nref} = 1.432 \cdot 10^{17} \left(\frac{Y}{300}\right)^{2.456} \quad \text{A-13}$$

$$N_{pref} = 2.67 \cdot 10^{17} \left(\frac{Y}{300}\right)^{2.456} \quad \text{A-14}$$

A.3 Field Dependent Mobility Model

The field dependent mobilities for n and p respectively are:

$$\mu_n = \frac{\mu_n^D}{\sqrt{1 + 1.54 \cdot 10^{-5} \cdot E}} \quad \text{A-15}$$

$$\mu_p = \frac{\mu_p^D}{\sqrt{1 + 5.35 \cdot 10^{-5} \cdot E}} \quad \text{A-16}$$

A.4 Sheet Resistance Calculation

After solving the Poisson equation, the sheet resistance for each semiconductor layer is estimated using:

$$R_{sh} = \frac{1}{\int_{xleft}^{xright} (q\mu_n \cdot n + q\mu_p \cdot p) dx} \quad \text{A-17}$$

$xleft$ and $xright$ are determined by the p - n junction locations and the semiconductor material boundaries.

A.5 Threshold Voltage Calculation

Threshold voltage calculation is based on the calculated sheet resistance. In MOS mode (1D vt extraction), the solver will calculate threshold voltage automatically. First, the conductance of the channel region will be calculated for each gate voltage applied. If an NMOSFET structure is assumed, then:

$$g(V_g) = \int_0^{x_{inv}} (q\mu_n)/ndx \quad A-18$$

0 corresponds to the oxide-silicon interface and x_{inv} is the boundary of the inversion layer. Threshold voltage will be determined using the $g(V_g)$ curve as an intersection with the V_g axis of the straight line drawn through two points on the $g(V_g)$ curve, corresponding to the maximum slope region shown below.

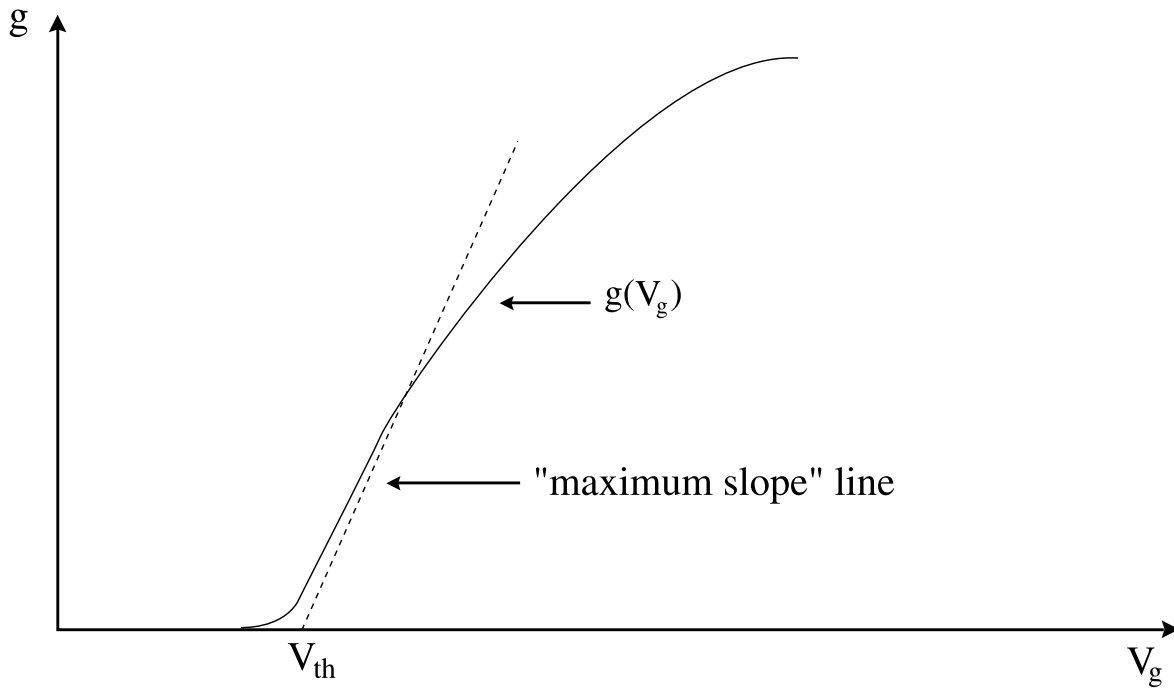


Figure A-1 Threshold Voltage Calculation

A.5.1 Breakdown Voltage Calculation

Breakdown voltage calculation is based on estimation of ionization integrals for electrons and holes. Breakdown is determined by the condition that one of the integrals is greater than 1. The ionization rates are calculated using the following equations (See the Selberherr model in the Atlas manual):

$$\alpha_n = AN \cdot \exp\left[-\left(\frac{BN}{E}\right)^{BETAN}\right] \quad A-19$$

$$\alpha_p = AP \cdot \exp\left[-\left(\frac{BP}{E}\right)^{BETAP}\right] \quad A-20$$

where:

AN = AN1 if E < EGRANAN = AN2 if E > EGRAN

AP = AP1 if E < EGRANAP = AP2 if E > EGRAN

BN = BN1 if E < EGRANBN = BN2 if E > EGRAN

BP = BP1 if E < EGRANBP = BP2 if E > EGRAN

The values of the parameters AN1, AN2, AP1, AP2, BN1, BN2, BP1, BP2, BETAN, BETAP, EGRAN are user-definable (through the `extract` command or pop-up menu). Their default values are:

AN1=7.03e5 cm⁻¹

AN2=7.03e5 cm⁻¹

BN1=1.231e6 V/cm

BN2=1.231e6 V/cm

AP1=6.71e5 cm⁻¹

AP2=1.582e6 cm⁻¹

BP1=1.693e6 V/cm

BP2=2.036e6 V/cm

BETAN=1.0 (unitless)

BETAP=1.0 (unitless)

EGRAN=4e5 V/cm



Appendix B

DBInternal

B.1 DBInternal

DBInternal is a simple but powerful DeckBuild tool that allows you to create a Design Of Experiments (DOE) from a pair of input files. Amongst other things, you can create corner models for process parameters or device characteristics or both.

Any parameters that are to be used as variables must be specified as `set` statements in a template file. Any results of interest should be calculated using `extract` statements.

The DOE is specified with simple `sweep` statements in a separate design file. The `sweep` statement defines which variables are required in the DOE, and the range of values these variables are to take.

The parameter values and the results of each simulation can be stored in a file that can be viewed in TonyPlot or used as a data base for input to a statistical analysis tool such as Spayn.

B.1.1 Example

Suppose you have an Atlas deck (`resistor_template.in`) for a simple resistor (the doping is controlled by a `set` statement). When run, this deck calculates the resistance from the gradient of the VI curve.

```
go atlas
set doping=1e16
set trial_id=0
mesh width=2
x.mesh loc=0.0 spac=0.25
x.mesh loc=1.0 spac=0.25
y.mesh loc=0.0 spac=0.25
y.mesh loc=10.0 spac=0.25
region num=1 silicon
electrode num=1 top name=ground
electrode num=2 bottom name=anode
doping uniform n.type conc=$doping
models conmob
solve init
log outf=dop$doping'.log
solve vanode=0.0 vstep=0.1 vfinal=2.0 name=anode
log off
extract init infile="dop$doping'.log"
extract name="res" grad from curve(i."anode",v."anode") where
y.val=1
quit
```

If you want to investigate how doping affects the resistance, you can create a DBInternal deck (`sweep.in`) that defines an experiment (a series of trials). In this example, the doping is changing between 10^{15} and 10^{19} cm^{-3} (at three points per decade, thirteen points in all).

```
go internal
load infile=resistor_template.in
sweep parameter=doping type=power range="1.0e15, 1.0e19, 13"
save type=sdb outfile=resistance.dat
quit
```

When you execute `sweep.in`, DBInternal runs the `resistor_template.in` deck several times, each time changing the value on the `set doping=` line. DBInternal also collates the data generated by the `extract name="res"` line and saves it in `sdb` format suitable for viewing with TonyPlot. The resistance as a function of doping is shown in the following figure.

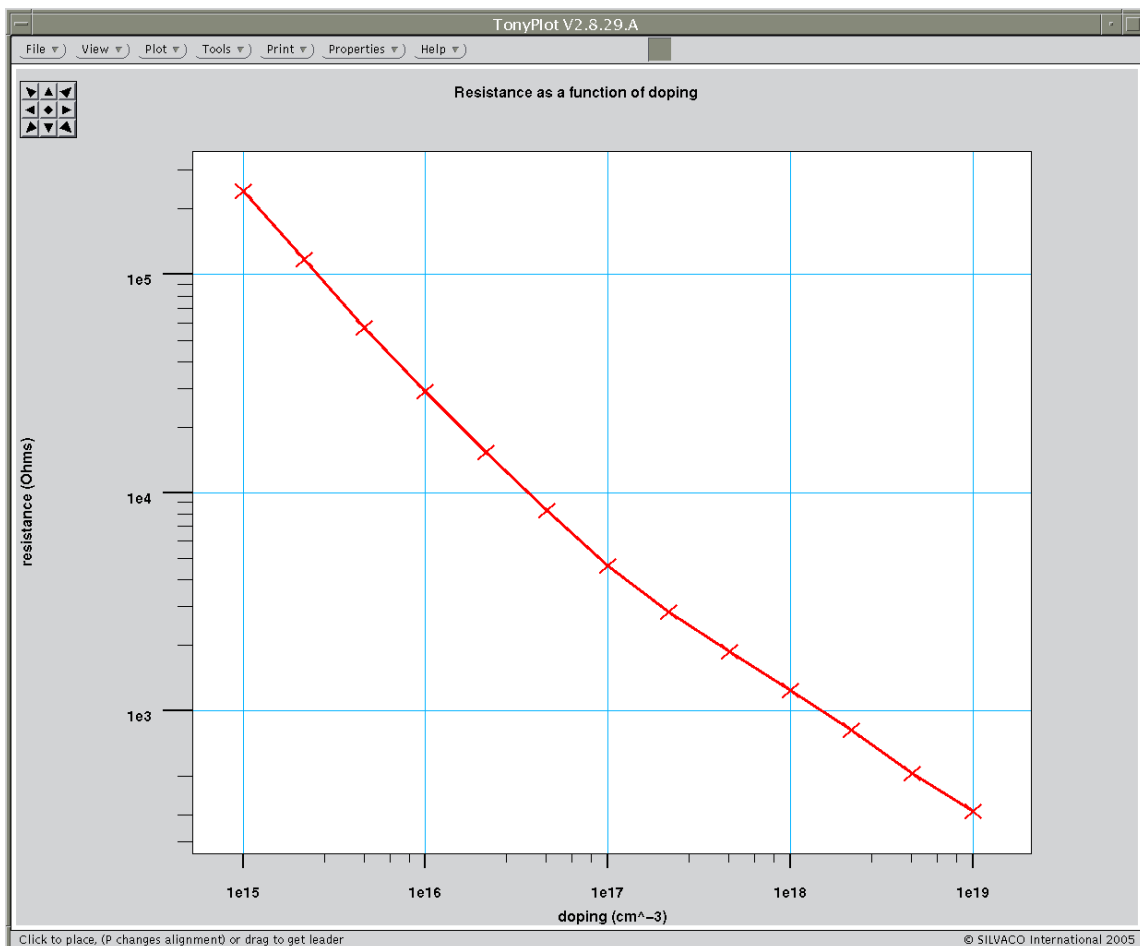


Figure B-1 Doping Resistance

B.2 The Template File

The template file is a description of the class of simulations you want to perform. It should be a deck that will execute correctly when run within DeckBuild.

Any variables that you need DBInternal to control must be defined on a `set` line. For example, the file `resistor_template.in` has the line

```
set doping=1e16
```

so DBInternal can change the value of the variable `doping`. DBInternal ignores the actual value on the `set` line in the template file. It is safe to set variables that DBInternal doesn't control. They will remain with the value defined in the template file.

The variable is normally used to set numbers in the template file. For example:

```
doping uniform n.type conc=$doping
```

where the doping concentration is being set by the variable `doping`. But it is also useful to be able to use the value as a string in a filename. In this instance, you should enclose the variable name in single quotes. For example:

```
log outf=dop$doping'.log
extract init infile="dop$doping'.log"
```

So if `doping` had been set to `1e16`, the filename would be `"dop1e16.log"`.

The template file may have `extract` statements. For example:

```
extract init infile="dop$doping'.log"
extract name="res" grad from curve(i."anode",v."anode") where
y.val=1
```

DBInternal will recognize you are interested in the result `"res"` (for example) and will collect these results from each simulation.

B.2.1 The `trial_id` Variable

The template file in [Section B.1.1 Example](#) contained the line

```
set trial_id=0
```

A variable called `trial_id` is used in a special way by DBInternal. If it exists in the template file, then DBInternal will assign the ordinal of the current trial to it during each child simulation.

In the previous example when the doping was `1e15` (the first trial), the `trial_id` variable would have been given a value of 1. When the doping was `1e16` (the fourth trial), the `trial_id` variable would have been given a value of 4. When the doping was `1e19` (the final trial), the `trial_id` variable would have been given a value of 13.

Because the `trial_id` is different for each trial, it is a useful way to generate a unique filename for each trial. In the example, we defined the log file with the command

```
log outf=dop$doping'.log
```

an alternative would have been

```
log outf=trial$'trial_id'.log
```

B.3 The Experiment File

The experiment file has three main parts

B.3.1 Load command

```
load infile=resistor_template.in
```

This tells DBInternal which file to use as the basis for the simulations.

B.3.2 Experiment command

```
sweep parameter=doping type=power range="1.0e15, 1.0e19, 13"
```

This tells DBInternal how you want the variables to change.

B.3.3 Save Command

If the template file contains `extract` statements, you also want a `save` command

```
save type=sdb outfile=resistance.dat
```

which tells DBInternal where to save the extracted data. The saved file will contain the values of all the independent variables (the variables defined in the `experiment` command) and the values of all the dependent variables (the variables calculated with `extract` statements).

B.4 Technical Details

DBInternal reads in the template file and looks for any variables defined on an `extract` line and makes a note of their names. The name must be the first parameter after the `extract` command and have no spaces.

```
extract name="res" ...
```

DBInternal also knows what parameters have been set on the experiment line. For example, DBInternal will control the parameters `x` and `y`.

```
sweep parameter=x type=linear range=1,2,2 \  
      parameter=y type=linear range=3,5,3
```

To run a trial, DBInternal creates a temporary `<infile>` with the name

```
<infile>=dbinternal_temporary_<name>_<pid>
```

`<name>` is the name of the machine and `<pid>` is the program ID of the DBInternal program. (See also the `LOG` command). The temporary file is a copy of the template file with different values on any set line that correspond to a parameter in the experiment command. For instance, if the template file had the lines

```
set x=5  
set y=10  
set z=15
```

and you ran the earlier `sweep` command the first temporary file would have the lines

```
set x=1  
set y=3  
set z=15
```

DBInternal will change the values for parameters it recognizes and leaves the other `set` lines alone.

DBInternal then runs a trial by producing a child (DeckBuild) with the command

```
deckbuild [-v a.b.c.X] [-int] [-ascii] [-noplot] -run <infile> -out-  
file  
<infile>.out
```

Once DeckBuild is finished, DBInternal will parse the `<infile>.out` file to find the values generated by the `extract` statements. The `<infile>` and the `<infile>.out` are then deleted.

This procedure (creating an `<infile>`, starting DeckBuild, examining the `<infile>.out`) is repeated for the remaining sets of parameters in the experiment.

B.5 DBInternal Commands

DBInternal commands are generally of the form

```
<command> <param1>=<value1> <param2>=<value2> ...
```

The commands and the parameters may be abbreviated but they must be long enough to be recognized. For instance, the `save` command may be shortened to `sa`, but not to `s` because that would not distinguish between `save` and `sweep`.

If the value contains whitespace, it must be enclosed in quotes. For instance

```
range="1.0e15, 1.0e19, 13"
```

But if the value is a single block of text, there is no need for the quotes. This range can be entered as

```
range=1.0e15,1.0e19,13
```

DBInternal recognizes the following commands.

B.5.1 convert

Syntax

```
convert <number> <string> into <string>
```

Description

The `convert` command converts a number from one set of units into another. This command tries to parse anything between `convert` and `into` into a value with a unit. And, it parses anything after `into` into the units to convert the value into.

Example

```
convert 1 eV into J
```

gives

```
1 eV == 1.60218e-19 J
```

B.5.2 doe

Syntax

```
doe type=<doe_type> \  
    parameter=<param1> range="center, delta" \  
    parameter=<param2> range="center, delta"
```

Description

The trials of a DOE experiment correspond to various points on or near a hypercube around some origin in parameter space. The results can be used to create a model for the dependent variables over the hypercube.

For example, suppose you had a process that generated FETs with gate lengths in the range 95-105 μm and recess depths in the range 45-55 μm . The parameter space is a square with corners (95, 45), (95, 55), (105, 45) and (105, 55). Suppose you want to know how breakdown voltage is affected by these parameters. If you knew there were a simple linear relationship, you could simulate at the midpoint and the two points where the axis intersected with the hypercube ((100, 50), (105, 50), (100, 55)) and fit a simple linear model through the

results. If you thought there were a more complex relationship, you would simulate at more points over the hypercube. For example, the midpoint and all the corners.

A parameter should be the name of a variable in the template deck. The names (e.g., <param1>) cannot be abbreviated. They must be exactly as they appear in the template deck.

The range is two numbers. The first is center of the hypercube (i.e., the value of the parameter at the middle of its range). The second is the distance from the center to the edge of the hypercube (or half the range of the parameter).

The points are almost always high symmetry points on the hypercube, such as the corners of the hypercube, the points where an axis intersects the hypercube, and a midpoint along an edge of the hypercube. The best way to see the points generated by a DOE type is to use the `no_exec` command and setting the range of the parameters to "0, 1". Therefore, 0 indicates a center point, 1 indicates one side of the hypercube and -1 the other side.

Example

```
no_exec outfile=tlff.dat
doe type=two_level_full_factorial \
    parameter=p1 range=0,1 \
    parameter=p2 range=0,1 \
    parameter=p3 range=0,1
```

DOE Types

The DOE type must one of the following:

- `gradient_analysis`
- `two_level_full_factorial`
- `two_level_half_factorial`
- `three_level_full_factorial`
- `face_centered_cubic`
- `circumscribed_circle`
- `box_behnken`

gradient_analysis

This type does a simulation at the center point and at the points one positive step along each axis. An experiment with N parameters has $N+1$ trials.

two_level_full_factorial

This type does a simulation corresponding to every node of the N -dimensional hypercube. An experiment with N parameters has 2^N trials.

two_level_half_factorial

This type does half the simulations of the `two_level_full_factorial` and no two nodes are on the same edge. An experiment with N parameters has 2^{N-1} trials.

three_level_full_factorial

This type does a simulation corresponding to every node and every half point of the N -dimensional hypercube. An experiment with N parameters has 3^N trials.

face_centered_cubic

This type does a simulation corresponding to every node. Every point where an axis intersects the hypercube and the center point. An experiment with N parameters has $2^N + 2N + 1$ trials.

circumscribed_circle

This type is similar to the `face_centered_cubic` type but the axis points now lie on the surface of the hypersphere that passes through the hypercube node points (i.e., all points are equidistant from the origin). An experiment with N parameters has $2^N + 2N + 1$ trials.

box_behnken

The design matrix for this simulation is based on a balanced or partially balanced block design. There is no easy relationship between the number of parameters and the number of trials in the experiment. For example, an experiment with seven parameters requires 57 trials and an experiment with nine parameters requires 97 trials. But an experiment with eight parameters is particularly inefficient and requires 225 trials.

B.5.3 endsave

Syntax

```
endsave
```

Description

This command tells DBInternal to stop saving data to the current file. See the `save` command for more information.

Example

```
save type=sdb outfile=example.out
sweep parameter=doping type=power range=1e15,1e19,13
endsave
```

This stops DBInternal from trying to save any more data to `example.out`.

B.5.4 get_data

Syntax

```
get_data infile=<filename> outfile=<filename> \
name="<string>" [ssv,tsv,csv] [[!]header] [[!]sort]
```

Description

`infile` is the name of the `.log` or `.str` file to read in.

`outfile` is the name of the file to save the extracted data to.

`name` is the name of the data to be extracted from the input file; the `;` character is used to separate different columns of data.

The name of the data is the name it is displayed under in TonyPlot. When extracting data from `.str` files, X, Y, and Z should be used to indicate the coordinate is required.

The data can be output in space-separated value (`ssv`), tab-separated value (`tsv`), or comma-separated value (`csv`) format. Specify `ssv`, `tsv`, or `csv` on the command to get the required format. `ssv` is the default if nothing is supplied.

The first row of the output file is usually the header information for each column. If you don't want this then put `!header` on the command.

The data is usually output in the order it is extracted from the input file. If you need to sort the data, then use `sort` on the command. This sorts the data in numerical order in the first column. If several rows have the same value in the first column, they are sorted in numerical order in the second column and so on.

Example

```
get_data infile=dciv.log outfile=gd1.dat \  
name="anode voltage; anode current"
```

This gets the anode voltage and anode current from a DC-IV curve.

```
get_data infile=val.str outfile=gd5.dat \  
name="X;Y;Potential;Singlet Exciton Density;Electron QFL"
```

This gets the potential, singlet exciton density, and electron quasi-Fermi level, as a function of position, from a structure file.

B.5.5 log

Syntax

```
log outfile=<filename_root>
```

Description

This command tells DBInternal to keep the output generated by the child DeckBuild for each trial. When generating the temporary input file for a trial, DBInternal uses the `<filename_root>` and appends the ID of the trial (for example, the first trial has an ID of zero and the second trial has an ID of one). The usual command is issued to run the trial

```
deckbuild -int -noplot -run <infile> -outfile <infile>.out
```

At the end of the trial, only the `<infile>` is deleted and the `<infile>.out` will remain.

Example

```
load infile=example.in  
log outfile=keep  
sweep parameter=doping type=linear range=1,4,4
```

This will generate the files `keep1.out`, `keep2.out`, `keep3.out`, and `keep4.out`.

B.5.6 monte_carlo

Syntax

```
monte_carlo number=<num_trials> \  
    parameter=<param1> type=<mc_type> coeffs="list, of,  
    numbers" \  
    parameter=<param2> type=<mc_type> coeffs="list, of,  
    numbers"
```

Description

The `monte_carlo` command generates an experiment from a specified number of trials. All parameter values in each trial are random. Each parameter is drawn from its own distribution.

The number is the number of trials you want in the experiment.

A parameter should be the name of a variable in the template deck. The names (e.g., `<param1>`) cannot be abbreviated. They must be exactly as they appear in the template deck.

The type must be one of the following:

- `uniform`
- `normal`
- `log_normal`
- `gamma`
- `weibull`

These are the types of random distributions a parameter will be extracted from. The coeffs are a list of numbers that describe the random distribution, the number and meaning of the coeffs depending upon which distribution was chosen.

Random Distributions

The probability density function and the required coefficients for the following random distributions.

uniform

The `uniform` type takes random numbers evenly spaced between two limits. The probability density function is

$$p(x) = 0 \quad (x < x_{lo} \text{ or } x \geq x_{hi})$$

$$p(x) = \frac{1}{x_{hi} - x_{lo}} \quad (x_{lo} \leq x < x_{hi}) \quad \text{B-1}$$

This distribution needs two coefficients, the minimum and maximum allowed values

$$coeffs = "x_{lo}, x_{hi}" \quad \text{B-2}$$

normal

The `normal` type is a Gaussian probability density. The probability density function is

$$p(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right) \quad \text{B-3}$$

This distribution needs two coefficients, the mean and the standard deviation.

$$\text{coeffs} = "\mu, \sigma" \quad \text{B-4}$$

log_normal

The `log_normal` type has a probability density function of (the distribution of $\log(x)$ would be normal)

$$p(x) = \frac{1}{\sqrt{2\pi} \cdot X\sigma} \exp\left(-\frac{(\log(X) - \mu)^2}{2\sigma^2}\right) \quad \text{with} \quad X = \pm(x - \tau) \quad \text{B-5}$$

This distribution needs four coefficients (the last coefficient should be +1 or -1 and gives the sign in front of X).

$$\text{coeffs} = "a \mu, \sigma, \pm" \quad \text{B-6}$$

gamma

The `gamma` type is the time to wait for several events that occur with a Poisson distribution. The probability density function is

$$p(x) = X^{a-1} \cdot \left(\frac{\exp(X)}{\sigma \cdot \Gamma(a)}\right) \quad \text{with} \quad X = \pm\left(\frac{x - \mu}{\sigma}\right). \quad \text{B-7}$$

This distribution needs four coefficients (the first coefficient should be a +ve integer and the last should be +1 or -1).

$$\text{coeffs} = "a \mu, \sigma, \pm" \quad \text{B-8}$$

Weibull

The probability density function of the `weibull` type is

$$p(x) = \frac{a}{\sigma} \cdot X^{a-1} \cdot \exp(-X^a) \quad \text{with} \quad X = \pm\left(\frac{x - \mu}{\sigma}\right) \quad \text{B-9}$$

This distribution needs four coefficients (the last coefficient should be +1 or -1).

$$\text{coeffs} = "a \mu, \sigma, \pm" \quad \text{B-10}$$

B.5.7 no_exec

Syntax

```
no_exec type=<ssf|spayn> outfile=<filename>
```

Description

This is a debugging command. If it is issued before an experiment, then DBInternal will generate a file <filename>. This file contains a list of the independent variables, which would have been passed to the simulations. No actual trials will be performed. An *ssf* format file is a simple list of numbers suitable for viewing in a text editor. It can also be plotted in TonyPlot. A *spayn* format file can be analyzed with Spayn.

Example

If you run an experiment file

```
no_exec type=ssf outfile=example.out
sweep parameter=length type=linear range=1,2,2 \
      parameter=width type=linear range=1,3,3
```

the data in *example.out* would be

```
...
0 1 1
1 2 1
2 1 2
3 2 2
4 1 3
5 2 3
```

The first column is the ordinal of the trial. The second column is the length value of that trial (e.g., 1 or 2). The third column is the width value of that trial (e.g., 1, 2 or 3).

B.5.8 option

Syntax

```
option [[!]int] [[!]ascii] [[!]plot] [version=<string>]
[[!]ignore.return]
```

Description

The *option* command controls some of the command line options passed to DeckBuild when running a trial.

The value of a parameter (*int*, *ascii*, or *plot*) is either true or false. If the parameter is present, it is set to true. If it is negated (with the !), it is set to false. If it is absent, it is set to a default value. The default for *int* is true. The default for both *ascii* and *plot* is false.

The *version* parameter defines which version of DeckBuild to use to run the child process.

A trial is run with the command

```
deckbuild [-v a.b.c.X] [-int] [-ascii] [-noplots] -run <infile>
-outfile <infile>.out
```

The `-int` option tells DeckBuild to start DBInternal as the default simulator. The `-ascii` option tells DeckBuild not to start its GUI. The `-noplot` option tells DeckBuild to ignore any TonyPlot commands in `<infile>`.

The `-int` option will be passed if `int` is `true` and will be absent if `int` is `false`.

The `-ascii` option will be passed if `ascii` is `true` and will be absent if `ascii` is `false`.

The `-noplot` option will be passed if `plot` is `false` and will be absent if `plot` is `true`.

The `ignore.return` flag indicates that DBInternal should ignore the return value of a child simulation. Even if the child simulation indicates an error DBInternal will try to parse the output file for any data.

Example

```
option ascii !plot
```

This will run the trials without the DeckBuild GUI and ignoring any TonyPlot commands in the trial deck.

```
option !ascii plot version=3.22.4.C
```

This will run the trials, using version 3.22.4.C of DeckBuild, inside a child DeckBuild GUI and will execute any TonyPlot commands in the trial deck.

B.5.9 save

Syntax

```
save    type=<sdb|spayn>    outfile=<filename>    [all.bad.skip]
        [any.bad.skip]
```

Description

The `save` command saves the data generated by the experiment in the file `<filename>`. You can output the data in `sdb` format (to be viewed in TonyPlot) or in `spayn` format (to be analyzed by Spayn).

The following data is stored for each trial:

- The ID of the trial.
- The values of the parameters defined on the experiment line.
- The values of the parameters calculated with `extract` commands.

You can place the `save` command before or after the experiment line. If it comes before the experiment line, the file will be rewritten at the end of each trial. Therefore if something unforeseen happens during an experiment, you will have the data from the trials that were completed. If it comes after the experiment line, all the data from the experiment will be written at once.

Only one file at a time can be active. If you define two `save` statements before an experiment line, only the second will actually get the following data.

```
save type=sdb outfile=save.sdb
save type=spayn outfile=save.spayn
sweep parameter=doping type=power range=1e15,1e19,13
```

The file `save.sdb` will have no data (the file `save.spayn` will be fine). Once a file is active, it will remain active until a subsequent `save` command makes a different file active or until an `endsave` command is given. A file, however, will only have data from one experiment.

Warning: If you have more than one experiment line in a deck, be very careful with the `save` command or you will lose data. For example, the following is the wrong way.

```
sweep parameter=doping type=power range=1e15,1e19,13
save type=sdb outfile=one.sdb
sweep parameter=doping type=linear range=1e16,1e17,11
save type=sdb outfile=two.sdb
```

This will perform the first experiment, then save that experiment to `one.sdb`, then perform the second experiment. But because `one.sdb` is still the active file, DBInternal will write the data from the second experiment to `one.sdb`, destroying the data that was already there. At the end of this run, both `one.sdb` and `two.sdb` will contain the same data. In this case, you must use the `endsave` command to tell DBInternal to deactivate the active file.

It is possible that an `EXTRACT` statement will fail to calculate a value. For example, an `EXTRACT` statement such as

```
EXTRACT name=vi01 y.val from curve(i."anode", v."anode")
```

where `x.val=0.1` will fail if an anode current of 0.1 A doesn't exist in the IV curve. A more extreme reason for failure would be if the simulation crashes before reaching the `EXTRACT` line.

The `all.bad.skip` and `any.bad.skip` parameters control when to save trials where `EXTRACT` has failed to calculate a value. If `all.bad.skip` is set, then any trials where all `EXTRACT` lines fail to calculate a value will not be saved to the output file. If `any.bad.skip` is set, then any trials where one-or-more `EXTRACT` lines fail to calculate a value will not be saved. If there are no `EXTRACT` lines in the template file and you are just saving the set values, then these parameters will be ignored and all lines will be saved.

Example

```
sweep parameter=doping type=power range=1e15,1e19,13
save type=sdb outfile=one.sdb
endsave
sweep parameter=doping type=linear range=1e16,1e17,11
save type=sdb outfile=two.sdb
```


B.5.10 sweep

Syntax

```
sweep parameter=<param1> type=<linear|power> range="start, stop,
num" \
    parameter=<param2> type=<list> data="list, of, ..., points" \
    [linked=<param3> type=<sweep_type> range="range"]
```

Description

The `sweep` command generates an experiment from all combinations of individual parameter values. The first parameter changes with the highest frequency. The final parameter changes with the lowest frequency.

A parameter should be the name of a variable in the template deck. The names (e.g., `<param1>`) cannot be abbreviated. They must be exactly as they appear in the template deck.

The type must be either `linear`, `power`, or `list`. If the type is `linear` or `power`, then the range should be three numbers. The first number is the initial value of the parameter, the second number is the final value of the parameter, and the third number is the number of points. If the type is `"list"`, then the data is just the list of values that should be assigned to the parameter.

Note: `data` is just a synonym for `range` so either name will work.

Any parameter defined with the `parameter` command varies independently with all other parameters defined with the `parameter` command. For example

```
sweep parameter=x type=linear range="1,3,3" \
    parameter=y type=linear range="10,30,3"
```

The `x` parameter is given the values 1, 2, 3 and the `y` parameter is given the values 10, 20, 30. This command would generate an experiment with 9 trials where each `x` value is paired with all `y` values: (1, 10), (2, 10), (3, 10), (1, 20), (2, 20), (3, 20), (1, 30), (2, 30), (3, 30).

But any parameter defined with the `linked` command is tied to the previous parameter. For example

```
sweep parameter=x type=linear range="1,3,3" \
    linked=y type=linear range="10,30,3"
```

(where `y` is now linked to `x`) would generate an experiment with 3 trials where `"x"` and `"y"` vary together: (1,10), (2,20), (3,30).

The order of these commands is important with any particular `linked` variable tied to the immediately previous variable. For example

```
sweep parameter=A type=linear range="1,3,3" \
    linked=B type=linear range="1,3,3" \
    parameter=C type=linear range="1,4,4" \
    parameter=D type=linear range="1,5,5" \
    linked=E type=linear range="1,5,5" \
    linked=F type=linear range="1,5,5"
```

Here, B is linked to A; E and F are linked to D.

The number of steps is defined by the parameter variable. For example

```
sweep parameter=A type=linear range="1,3,3" \  
    linked=B type=linear range="1,2,2" \  
    linked=C type=linear range="1,4,4"
```

A is defined to have the values (1, 2, 3). B is defined to have the values (1, 2). C is defined to have the values (1, 2, 3, 4). However, the experiment is controlled by the parameter A and will contain three trials: (1, 1, 1), (2, 2, 2), (3, 2, 3). B reaches its final value of 2 and then stays there and the final value of C is never reached.

Example 1

In a linear sweep, the parameter values are evenly spaced.

```
sweep parameter=x type=linear range="1,4,7"
```

This generates for x the values:

- 1
- 1.5
- 2
- 2.5
- 3
- 3.5
- 4

Example 2

In a power sweep, the log of the parameter values are evenly spaced.

```
sweep parameter=y type=power range="1e10, 1e15, 6"
```

This generates for y the values:

- 1e10
- 1e11
- 1e12
- 1e13
- 1e14
- 1e15

Example 3

The data is a list of values to assign to the parameter.

```
sweep parameter=z type=list data="1,2.5,3,3.1,4"
```

This assigns each of the values (one at a time) to z.

Example 4

The number of trials in the experiment is the product of the number of points for each independent parameter.

```
sweep parameter=x type=linear range="1,4,7" \  
      parameter=y type=power range="1e10, 1e15, 6"
```

This generates an experiment with 42 trials. All the values of x in combination with all the values of y. Adding another variable:

```
sweep parameter=x type=linear range="1,4,7" \  
      parameter=y type=power range="1e10, 1e15, 6" \  
      parameter=z type=list data="0, 2, 3"
```

would increase the number of trials to 126. All 42 trials from the previous experiment with z=0, and all 42 trials with z=2, and all 42 trials with z=3.

Example 5

Linked variable do not increase the number of trials in an experiment.

```
sweep parameter=temp type=linear range="200,400,5" \  
      linked=conc type=power range="1e15,1e19,5" \  
      linked=height type=list data="1,1.1,1.3,1.6,2"
```

This generates an experiment with 5 trials with the values for temp, conc, and height.

```
200, 1e15, 1.0  
250, 1e16, 1.1  
300, 1e17, 1.3  
350, 1e18, 1.6  
400, 1e19, 2.0
```

B.5.11 translate.ise

Syntax

```
translate.ise devedit|atlas|tonyplot \  
[bnd.file=<filename>] [cmd.file=<filename>] \  
[plt.file=<filename>] [grd.file=<filename>] \  
[dat.file=<filename>] \  
[out.file=<filename>] [pure.ac] [[!]execute]
```

Description

This command translates ISE™ decks and data files into Silvaco decks and data files. There are three versions of this command:

```
translate.ise devedit bnd.file=<filename> cmd.file<filename> \  
[out.file=<filename>] [[!]execute]
```

translates between MDraw™ and DevEdit decks.

```
translate.ise atlas cmd.file=<filename> \  
[out.file=<filename>] [pure.ac] [[!]execute]
```

translates between Dessis™ and Atlas decks.

```
translate.ise tonyplot [plt.file=<filename>] \  
[grd.file=<filename>] [dat.file=<filename>] \  
out.file=<filename> [[!]execute]
```

translates ISE™ data files so that they can be viewed in TonyPlot.

The `out.file` parameter is common to all versions of the command. This gives the name of the translated deck or data file that is created by the command. This is a required parameter for the `translate.ise tonyplot` command but is optional for the other two. If this parameter is not present, then the translated deck is written to the file `pp<n>_ded.cmd` or `pp<n>_atl.cmd` (where `<n>` is an integer).

The `execute` parameter is also common to all versions of the command. When the translation has been done, the appropriate Silvaco tool is run (DevEdit or Atlas are run in a child DeckBuild, plot files are shown in TonyPlot). If you just want the translation without running the Silvaco tool, then add `!execute` to the command.

The `pure.ac` parameter for the `translate.ise atlas` command tells the translator that the external circuit is just for a small-signal AC simulation and can be ignored.

The `translate.ise tonyplot` command can operate on a `plt.file` (which are 1D data like Silvaco's DC-IV `.log` files) or on a `grd.file/dat.file` pair (which are 2D data like Silvaco's device `.str` files).

B.6 DBIT

DBIT is a GUI tool that provides some of the functionality of DBInternal. DBIT runs outside of DeckBuild. But DeckBuild is still required to run the child trials. To start DBIT, type "dbit" at the command line. The DBIT Main Window will then appear.

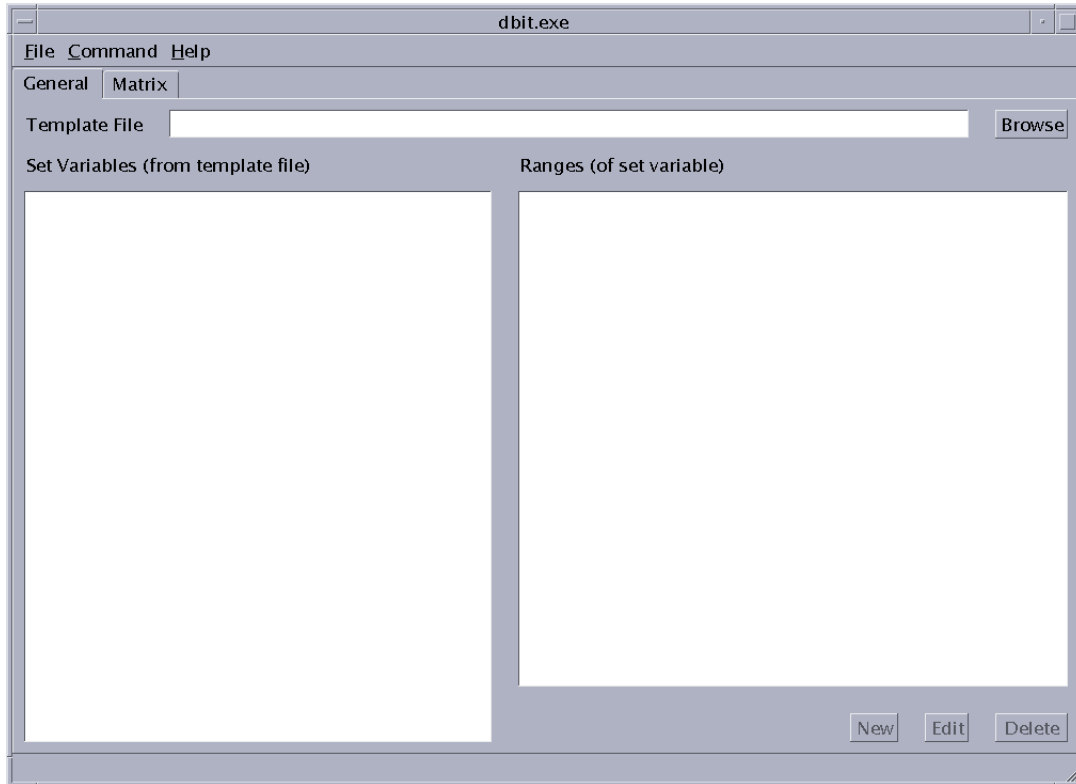


Figure B-2 DBIT Main Window

B.6.1 The General Tab

Click on the **Browse** button or select **File**→**Open** to open a template file. This file will be analyzed and the variables that appear on the "set" lines of the template file will be listed in the left hand box (see [Figure B-3](#)).

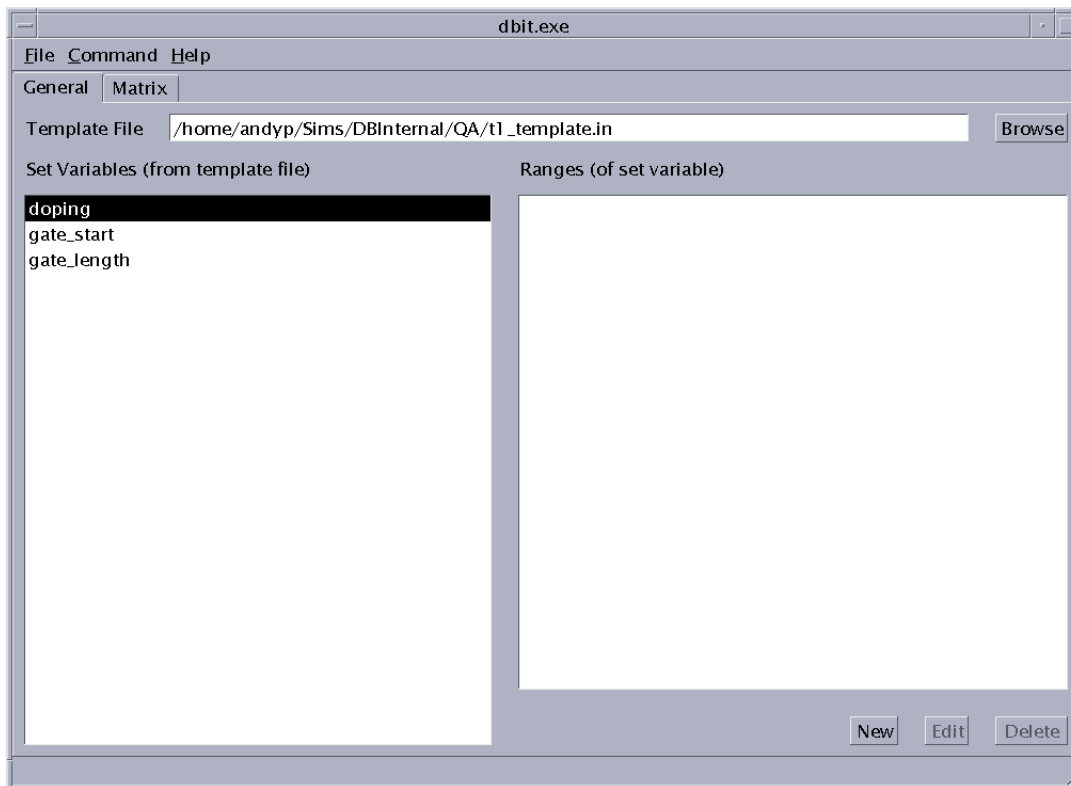


Figure B-3 General Tab

Note: The input file must be a template file rather than a dbinternal experiment file.

To define a range of values for a variable, highlight the appropriate variable in the left hand box and click the **New** button or double click the name of the variable. The Range Dialog Window will appear (see [Figure B-4](#)).

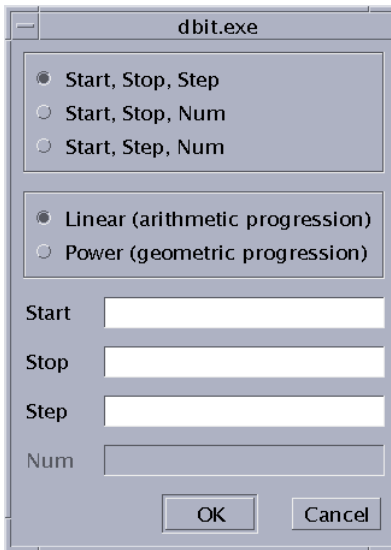


Figure B-4 Range Dialog Window

The range can be defined as an arithmetic progression (Equation B-11) or a geometric progression (Equation B-12).

$$a_1, a_1 + d, a_1 + 2 \cdot d, a_1 + 3 \cdot d, \dots, a_n = a_1 + (n - 1) \cdot d \quad \text{B-11}$$

$$a_1, a_1 \cdot r, a_1 \cdot r^2, a_1 \cdot r^3, \dots, a_n = a_1 \cdot r^{(n-1)} \quad \text{B-12}$$

There are four parameters associated with a range:

- the initial value (a_1)
- the number of points (n)
- the step (d for arithmetic progression and r for geometric progression)
- the final value (a_n).

A range can, in principal, be uniquely defined by giving any three of these parameters. But DBIT expects you always to define the initial value, so there are three ways to define a range:

- the initial value, the final value, and the step
- the initial value, the final value, and the number of points
- the initial value, the step, and the number of points

If you require a single value just enter it into Start.

If you use Start, Stop, Step to define the range, the final value of the range may not be the value explicitly written in Stop. If the progression defined by Start and Step does not exactly reach the Stop value, then the range will stop before the Stop value. For example, Start=1, Stop=10, Step=2 will result in the range "1,3,5,7,9" and not "1,3,5,7,9,10".

Once you add a range, you can edit the description of the range by highlighting the range (in the right hand box) and clicking the **Edit** button (or by double clicking on the appropriate range). You can delete a range by highlighting it and clicking the **Delete** button.

B.6.2 The Matrix Tab

Once you finished defining the ranges for the variables, click on the **Matrix** tab. This shows a spreadsheet-like area that lists all the trials to be done (see [Figure B-5](#)).

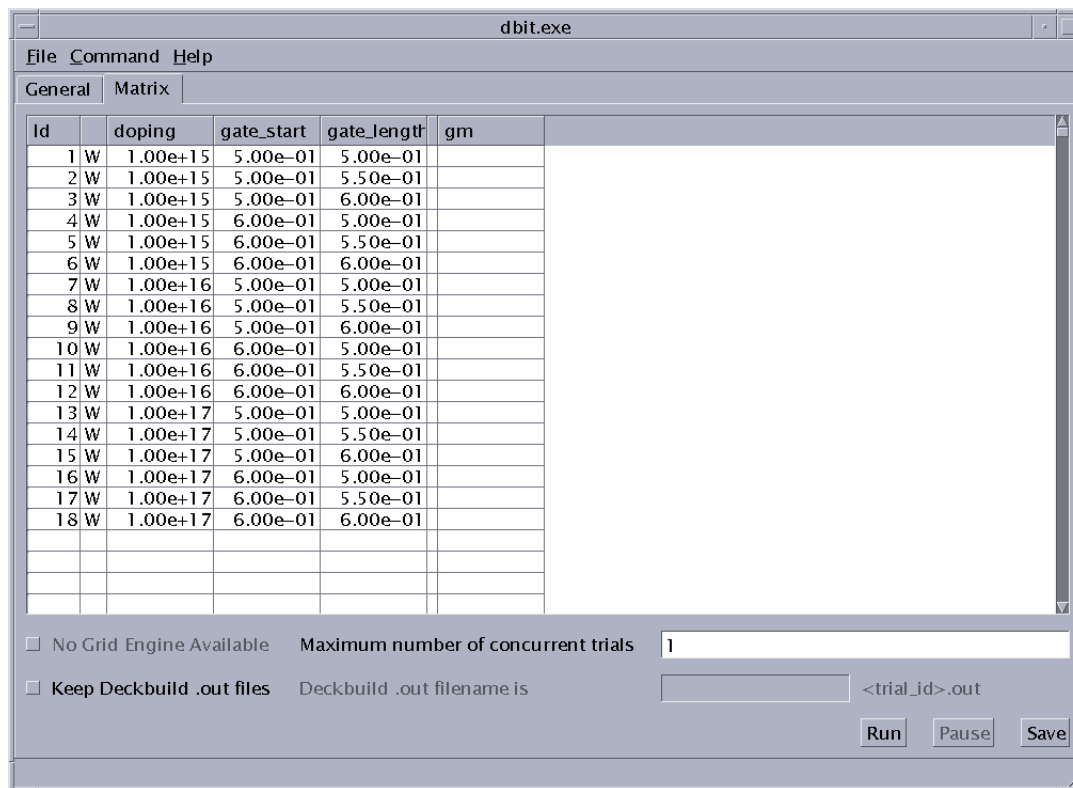


Figure B-5 Matrix Tab

Click the **Run** button to run the trials. The trials are run in the order they are defined in the spreadsheet. When one trial is finished, a new trial starts with the next available set of numbers.

If you click the **Pause** button, no new trials will be started (until you click on the **Run** button again) but any currently running trials will be allowed to finish.

Each row in the spreadsheet corresponds to a trial. You can edit various cells in the spreadsheet to change or add trials to the experiment. Clicking the right mouse button while the cursor is over the spreadsheet brings up a menu with the available commands.

The first column is **Id**. This is an integer that is assigned by DBIT to a trial. Each row will have a unique integer. If you add a trial to the list, it will automatically assign the next available Id. You cannot change the value in this column. The Id assigned to a trial is available within the template file by using `$trial_id`. For example, you can output data in the template file data with the command

```
save outfile=data_`trial_id`.dat
```

This will save the data to a different file for each trial.

The second column is a single character that gives the current status of the trial. A 'W' means the trial is waiting to run. An 'R' means the trial is currently running (or has finished running but the output files have not yet been processed). An 'F' means the trial finished successfully. An 'X' means the trial failed to start. An 'H' means you have hidden the trial.

A hidden trial is still displayed in the spreadsheet but DBIT will not run it. To hide a trial, select the rows that correspond to the trials you want to hide, click the right mouse button, and select **Hide Trial**. You can reactivate a hidden trial by selecting the **Activate Trial** option.

The next few columns in the spreadsheet correspond to the set variables, one column for each variable (the name of the variable is at the top of the column). The values in the columns are the values that will be assigned to the variables when the corresponding trial is running. You can edit values in these columns by selecting an appropriate cell and typing the new value. You can Cut, Copy and Paste values to and from these columns. If you add numbers to a blank row, a new trial will be generated. The next available trial ID will be automatically added to the first column.

A blank column comes next to mark the end of the set variables.

Any remaining columns correspond to the extract variables (the name of the variable is at the top of the column). These columns are initially empty. When a trial successfully runs, the results of the `Extract` commands will be added to the appropriate cells.

To save the data in the spreadsheet, either click the **Save** button or select **File→Save** or **File→Save As**. You can save the data in `.dat` format to view it in TonyPlot or as space-delimited text that can be imported into a spreadsheet program. Selecting **File→Save As** allows you to define the filename and the file format. Pressing the **Save** button or selecting **File→Save** saves the data to the most recent file defined by **File→Save As**. If you haven't selected **File→Save As**, then pressing the **Save** button or selecting **File→Save** opens the File→Save As Dialog.

The **Keep Deckbuild .out files** check box behaves in the same way as the `"log"` command for DBInternal. If you check this box, the `.out` files are not deleted when a trial is finished.

The **Maximum number of concurrent trials** defines the maximum number of child deckbuilds that DBIT can start. If you are running on a computer with more than one processor, or if you have a grid computing engine, you can set this higher than 1.

DBIT is designed to use Flowtracer/NC™ by Runtime Design Automation or the N1 Grid Engine™ by Sun. If DBIT detects a grid computing engine, you can run the trials on the network rather than the local machine.

B.6.3 The Command Menu

The **Command** menu gives you access to the commands that are used to run the child simulations. In these dialog boxes, the text in angular brackets (such as `<in_file>` and `<silock_command>`) are text generated by DBIT, which will be different for each trial. Leave these variables as they are.

“deckbuild...”

The default command to start the child simulation is

```
deckbuild "run <in_file> -int "outfile <out_file>
```

If you want to add any command line options to this command (see [Section B.4 Technical Details](#)), then add them in this dialog window.

“silock (local host)...” and “silock (grid engine)...”.

`silock` is a small program that monitors the progress of a child trial (started with the `deckbuild` command). When the trial is finished, `silock` removes a lock file that allows DBIT to determine the trial has finished. The two versions of the `silock` command are for when the trials are run directly by DBIT (the local host version) or when the trials are submitted to a grid engine. You should not change these commands. The default command for “`silock (local host)...`” is

```
silock "f "l <lock_file> -x <deckbuild_command>
```

and the default command for “`silock (grid engine)...`” is

```
silock "l <lock_file> -x <deckbuild_command>
```

“Flowtracer/NC...” and “Sun N1...”

These are the default commands used to submit a job to the appropriate grid engines.

The default command for Flowtracer/NC™ is

```
nc run -E 'SNAPSHOT+D(DISPLAY=[hostname]:0)' <silock_command>
```

The default command for N1 Grid Engine™ is

```
qsub -cwd -N <trial_name> -b y "<silock_command>"
```

If you have either of these grid engines, please refer to their documentation for more details on these commands.

A
Athena 9
Auto-Interface 9
B
Bipolar Extract
 QUICKBIP 205
BJT 196
Breakdown Voltage Calculation 224
C
Calculation
 Breakdown Voltage 224
 Sheet Resistance 222
 Threshold Voltage 223
Commands Menu 81
Concentration Dependent Mobility 220
Curves
 Abs Operator with Axis 198
 Ave Operator 197
 Axis Manipulation Combined with Max and Abs Operators 199
 Axis Manipulation Combined with Y Value Intercept 199
 Axis Manipulation with Constants 198
 Creation 197
 Data Format File Extract with X Limits 199
 Derivative 199
 Gradient at Axis Intercept 198
 Impurity Transform against Depth 199
 Max Operator 197
 Max Operator with Axis Intercept 198
 Min Operator 197
 Min Operator with Axis Intercept 198
 Second Intercept Occurrence 198
 X Axis Interception of Line Created by Maxslope Operator 199
 X Value Intercept for Specified Y 197
 Y Axis Interception of Line Created by Minslope Operator 199
 Y Value Intercept for Specified X 198
Customized Extract Statements
 Defaults 182–183
 Syntax 138–181
D
DBInternal 232
 Commands 232–243
 DBIT 245–250
 Example 227–228
 Experiment File 230
 Technical Details 231
 Template File 229
DBInternal Commands
 convert 232
 doe 232–234

- endsave 234
- get_data 234–235
- log 235
- monte_carlo 236–237
- no_exec 238
- option 238–239
- save 239–240
- sweep 241–243
- translate.isc 241
- DBIT 245–250
- Deck Parsing 81
- Deck Writing 81–87
- Design of Experiments (DOE)
 - box_behnken 234
 - circumscribed_circle 234
 - face_centered_cubic 234
 - gradient_analysis 233
 - three_level_full_factorial 233
 - two_level_full_factorial 233
 - two_level_half_factorial 233
- Dessis™ 244
- DevEdit 9
- Device Extraction 192–196
 - BJT 196
 - Curve 192
 - Curve Manipulation 194
- Device Simulation 9
- Diffusion Dialog 84
- E
- Examples 11
- Execution Control 10
- Extract 203
 - Customized Statements 138
 - Device Extraction 192
 - Features 203
 - MOS Device Tests 201
 - Process Extraction 131
 - QUICKBIP Bipolar Extract 205
 - Results 202
 - Using with ATLAS 208
- Extract Features
 - Extract Name 203
 - Extraction and the Database (VWF) 204
 - Min and Max Cutoff Values 204
 - Multi-Line Extract Statements 204
 - Variable Substitution 203
- Extraction 11

F
Field Dependent Mobility Model 221
Flowtracer/NC™ 249, 250
Full Interactive Control 9
Function Calculator 11
G
General Curve Examples 197–199
Generic Decks 11
H
History 10
I
Input Decks 9
Inputting Commands 81–87
Interactive Plotting 9
ISE™ 244
M
Mask
 Misalignment and CD Experimentation 119
MaskViews 11, 120
MDraw™ 244
MOS Device Tests 201
N
N1 Grid Engine™ 249, 250
O
Optimizer 9, 11
P
Physical Models 219
Process Extraction
 Curves 136
 Entering Statements 134
 Examples 183–191
Process Extraction Examples
 1D Material Region Boundary 186
 1D Max/Min Concentration 184
 2D Concentration Area 186
 2D Concentration File 185
 2D Material Region Boundary 186
 2D Max/Min Concentration 185
 2D Maximum Concentration File 186
 ED Tree (Optolith) 190
 Elapsed time 191
 Electrical Concentration Curve 190
 Junction Breakdown Curve 188
 Junction Capacitance Curve 187
 Junction Depth 183
 Material Thickness 183
 QUICKMOS 1D Vt 183
 QUICKMOS CV Curve 187
 Sheet Conductance 183

Sheet Resistance 183
Sheet Resistance/Conductance Bias Curves 189
SIMS Curve 189
SRP Curve 189
Surface Concentration 183
Process Input Deck
 Writing 84–85
Process Simulation 9
Process Simulators 83
Q
QUICKBIP
 Bipolar Extract 205–207
R
Random Distributions
 gamma 237
 log_normal 237
 normal 237
 uniform 236
 Weibull 237
S
Sheet Resistance Calculation 222
Simulators 9
SSuprem3 9
Statements
 ASSIGN 105–108
 AUTOELECTRODE 109
 DEFINE 110–111
 ELSE 115
 EXTRACT 112
 GO 113–114
 IF 115
 IF.END 115
 L.END 116–117
 L.MODIFY 116–117
 LOOP 116–117
 MASK 118–119
 MASKVIEWS 120
 SET 121–123
 SOURCE 124–125
 STMT 126
 SYSTEM 127
 TONYPLOT 128
 UNDEFINE 110–111
Stop Points 9
T
Text Editor 9
Threshold Voltage Calculation 223

trial_id 229
U
Utmost 9