VisualFab

Workbench

Version 1.0.0

VisualFab User's Guide:

Cogenda Pte Ltd

License Grant

Disclaimer

This documentation was typed in DocBook XML format, and typeset with the ConTEXt program. We sincerely thank the contributors of the two projects, for their excellent work as well as their generoisty.

Contents

CHAPTER 1

Introduction

Concepts

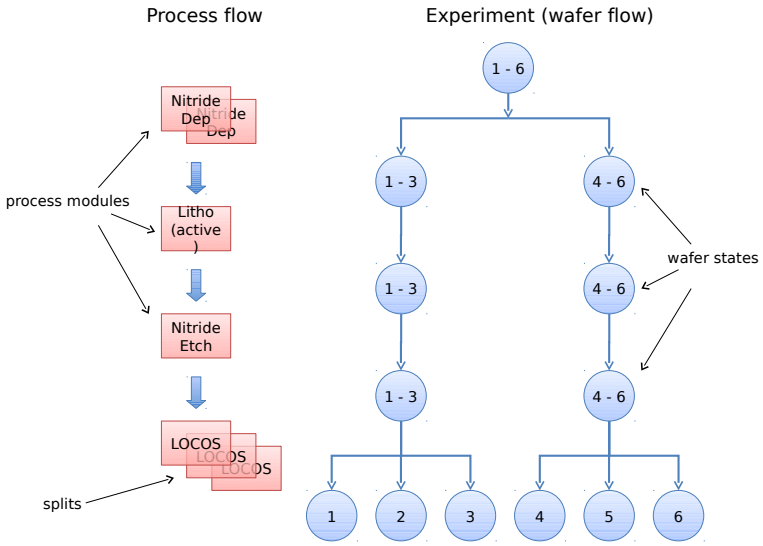

Figure 1.1 Real-world concepts v.s. VisualFab concepts

- Process step. This can be an ion implantation step, or a temperature ramp step in an annealing process. It is described by the name of the step and a series of parameters.
- Process module. This is a sequence of related process steps, e.g. "LOCOS isolation" or "LDD implant". This is basic unit of execution in VisualFab, i.e. all steps in the module are performed in the same simulation run. In VisualFab, the concept of process module is generalized to include device simulation, BSIM model extraction and other tasks.
- Process flow and Experiment. This is a sequence of process modules that produce a device or circuit.
  At some modules in the sequence, the user can introduce splits to some process parameters. This causes the process flow to branch at this module, and leading to a tree-like experiment structure with several process flow paths. This is analogous to a run-sheet.
- Wafer. Conceptually, silicon wafers are first placed at the beginning of the process flow. The wafers go through the process flow. When there are process splits, one wafer is needed for every possible path. Wafer States. Every process module does some change to a wafer, thus moving it to a new wafer state from

the parent state. Several wafer states can share a common parent state, at a module with multiple splits. Every wafer state is kept on disk, and can be examined at any time.

- Simulator. In VisualFab, a simulator can be a process simulator, a device simulator, a BSIM model extractor or in general any Unix program or script. Every module must assign a simulator.
- Inspector. User uses an inspector to examine wafer states. It can be an interactive visualization program for examining the device structure, or in general any Unix program. Unlike simulators, inspectors are not associated with a process module, thus not part of the process flow.
- Project. A project holds a collection of related experiments. Any object in the project can be re-used in several experiments.

CHAPTER 2

Getting Started

Installation

Starting VisualFab

To start VisualFab, one can type at the Linux command line:

```
$ /opt/cogenda/1.7.3/VisualTCAD/bin/VisualFab
```

In this tutorial guide, it is assumed that network machines and simulation tools have been properly setup. For details on network machines, simulators and inspectors, please see chapter xxx.

Basic Usage

The main tool-bar of VisualFab is shown in **Figure 2.1**, **p. 5**. Apart from the usual file and edit tools, there are buttons to active the network machine manager, and the workspace manager.

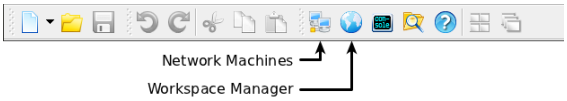

Figure 2.1 The VisualFab Main Tool-bar

Workspace Manager

The *workspace manager* shown in **Figure 2.2**, **p. 5** is usually the first window to open when one uses VisualFab. In the workspace manager, one can browse the projects and experiments in the workspace, open an experiment, create new projects or experiments, and copy/move experiments around.

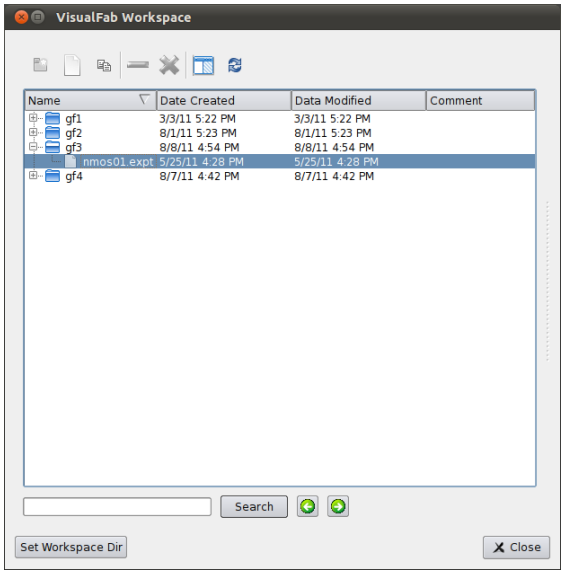

Figure 2.2 Workspace Manager

When VisualFab is used for the first time, the workspace is empty. One needs to set the directory to host workspace. In this tutorial, we assume that the workspace directory is `$HOME/vfab/`.

Each project is shown as a folder in the workspace, and experiments in the project listed in the folder. Project can be nested.

Editing an Experiment

We open the experiment `nmos01` in the workspace, and it's contents are displayed in the experiment window shown in **Figure 2.3, p. 6**. Three components are immediately visible.

1. Tool-bar (compact mode) for editing and running the experiment.
2. Process flow view for editing process modules.
3. Wafer states in the experiment.

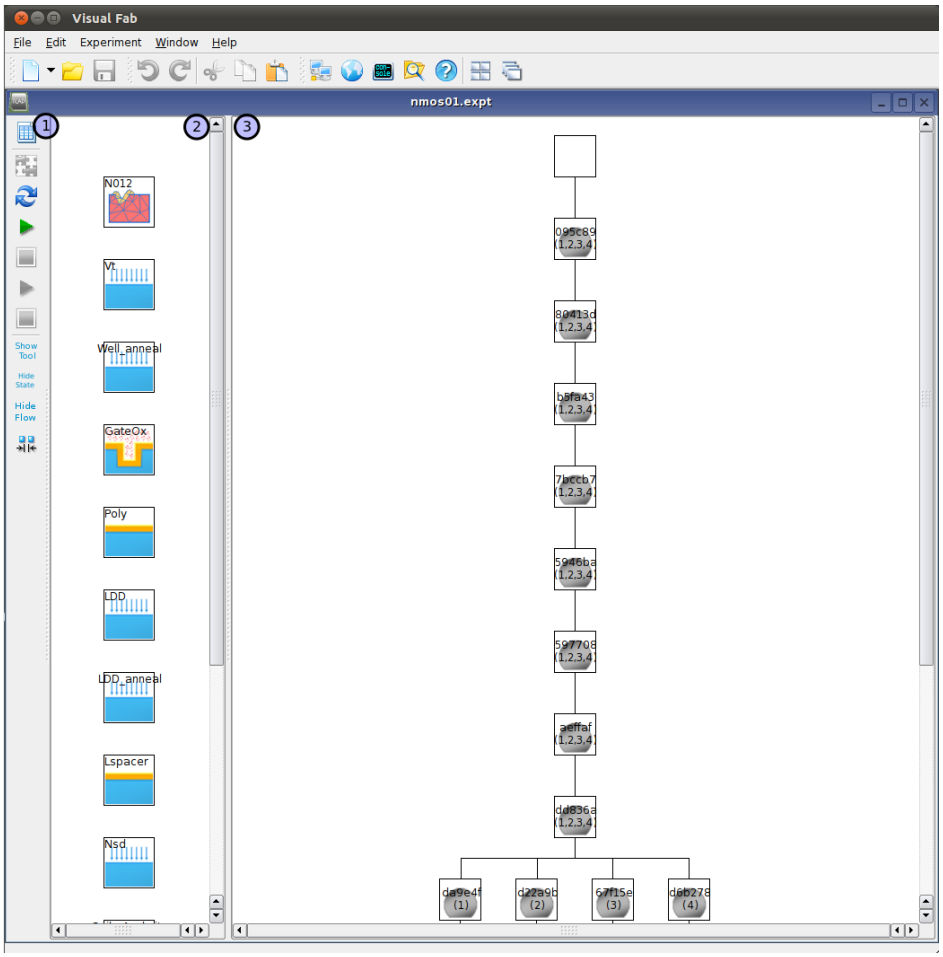

Figure 2.3 An nMOS experiment

The tool-bar is displayed in compact mode by default (**Figure 2.4, p. 7**).

In this section, we will use the run table button to examine the overall setup of the experiment, and run the experiment with the run button.

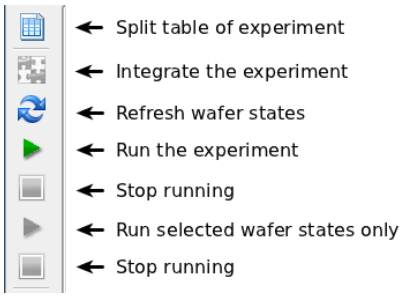

Figure 2.4 Tool-bar for an experiment.

Run Table

The *Run table* provides a summary of the experiment, containing all splits, control variables and output variables, as shown in Figure 2.5, p. 7.

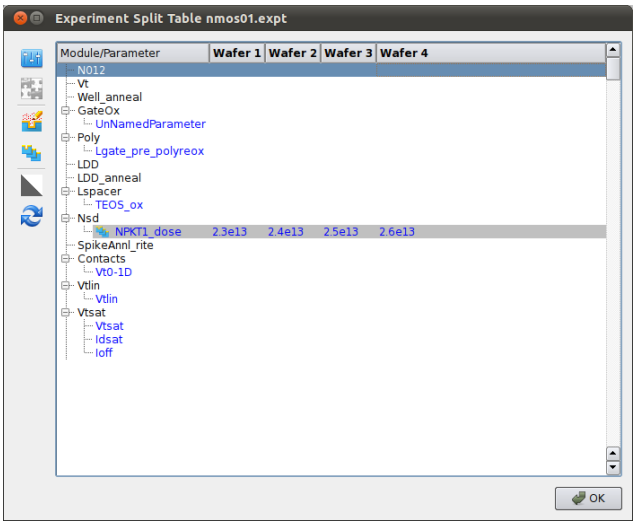

Figure 2.5 Run table of the NMOS experiment.

We can see that there are four splits in the *Nsd* module, on the `NPKT1_dose` parameter, which is marked with an icon. Four wafers will be produced by at the end of the experiment. Output variables are highlighted in blue color. Since the experiment is yet to be run, no output variable is available.

Running Simulation

In the tool-bar (Figure 2.4, p. 7), we click the Run Experiment button to start running.

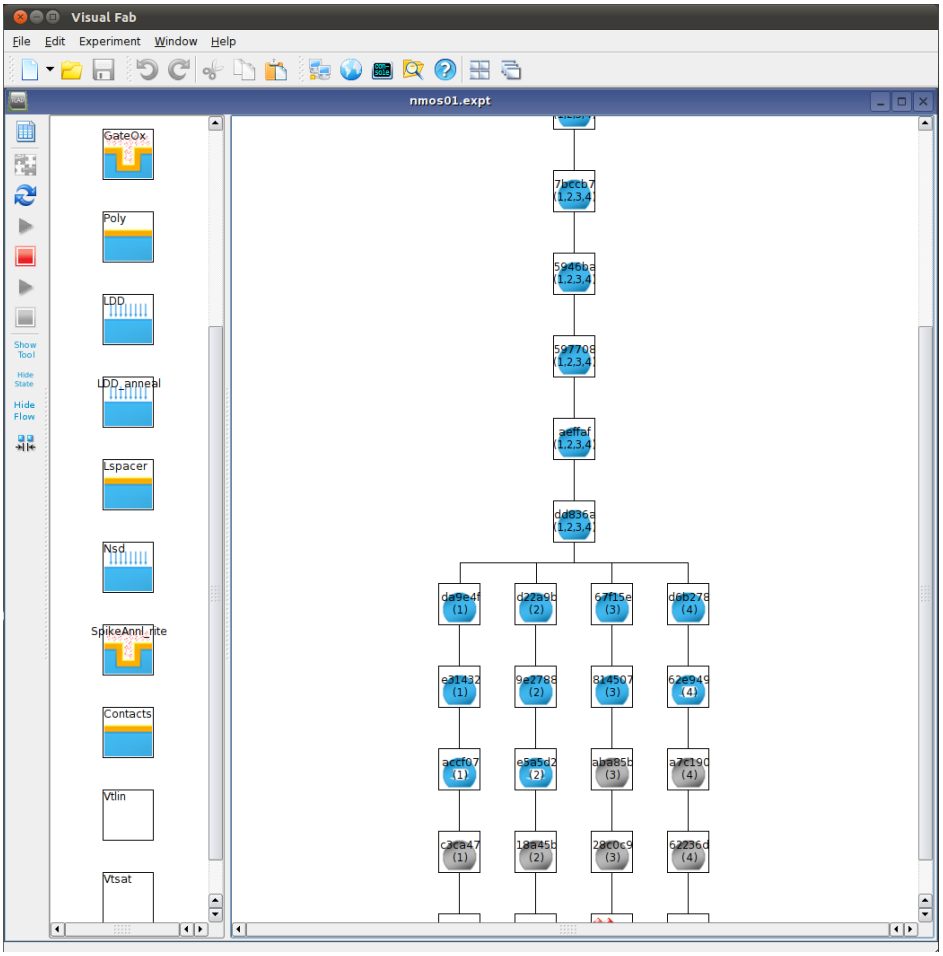

Figure 2.6 The experiment running.

If this button is disabled, and the wafer state view is empty, it's likely that the experiment's topology was changed, and needs to be re-integrated. The Integrate button does this.

When the experiment is running, one can stop it at any time, by clicking the Stop Experiment button.

It is also possible to select a few wafer states, right-click to get the context menu (Figure 2.7, p. 9), and choose only to run the selected wafer states. Similarly, one can selectively stop the running of some wafer states.

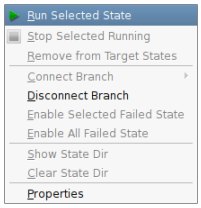

Figure 2.7 Run selected wafer states.

Checking Results

Inspectors

In order to view the simulated device structure, one first click the Show Tool button (**Figure 2.8**, **p. 10**) to see the list *Inspector* tools.

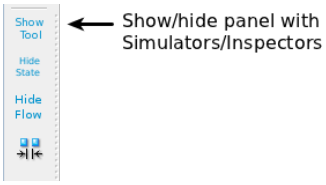

Figure 2.8 Showing the panel containing complete list of tools.

One can select a few wafer states, drag and drop them to the *TV2D* inspector. The TV2D program will be started, and the output device structure of each wafer state will be loaded and displayed.

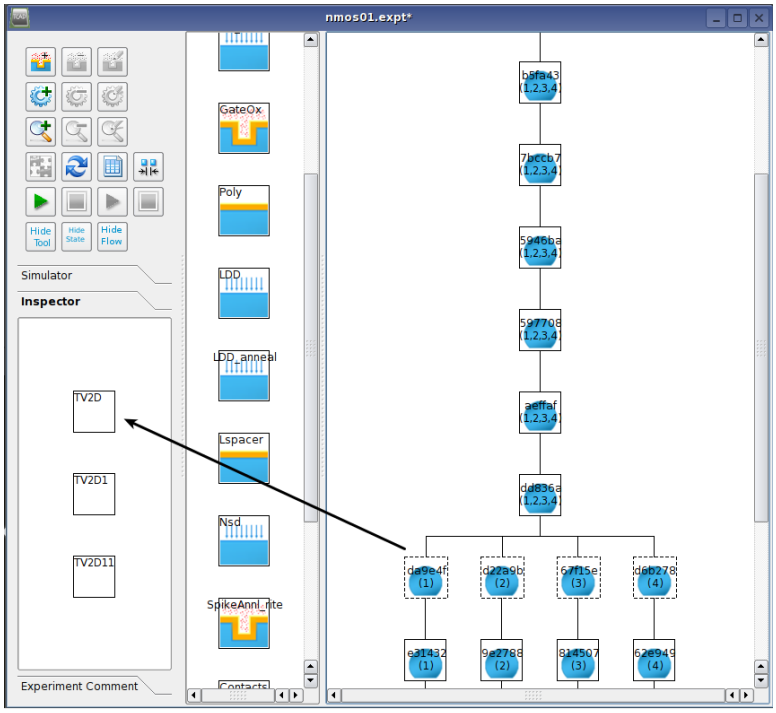

Figure 2.9 Viewing simulated device in a Inspector.

Alternatively, one can right-click on the wafer state, and select Begin Receive States. One can drop wafer states to the inspector, but the inspector is not started immediately. After all wafer states have been dropped to it, one can select End Receive States,

and only then the inspector will be started. With this mechanism, it is also possible to drag wafer states from several experiments, and open them in the same inspector.

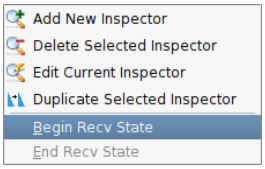

Figure 2.10 Inspector context-menu to begin/end receiving wafer states.

Run Table

After the experiment finished running, we can examine the extracted output variables in the run table again, as shown in **Figure 2.11**, **p. 11**.

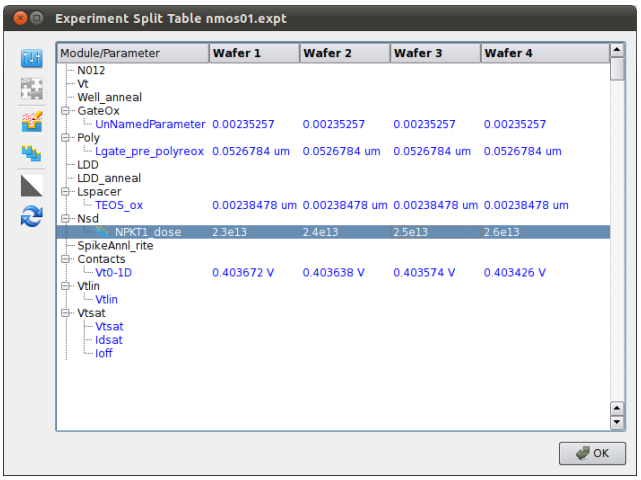

Figure 2.11 Run table of after simulation completes.

Editing Process Modules

The process flow consists of several process modules. To view and edit a process module, one can double-click it in the process flow view in Figure 2.3, p. 6. The process module dialog (Figure 2.12, p. 12) shows up with the following components:

1. Label. Each process module must have a distinctive name, Vt, Well_anneal, GateOx, etc. are some typical examples. Space and special characters are not allowed in label name.
2. Comment. Some textual description.
3. Symbol. A graphic symbol for the module.
4. Simulator. Select the process simulation tool that will be used to simulate this module.
5. Process Steps. The commands that will be passed to the tool for the simulation of his module.
6. Module Parameters. Some variables to be used in this module only. One can create split on module parameters.

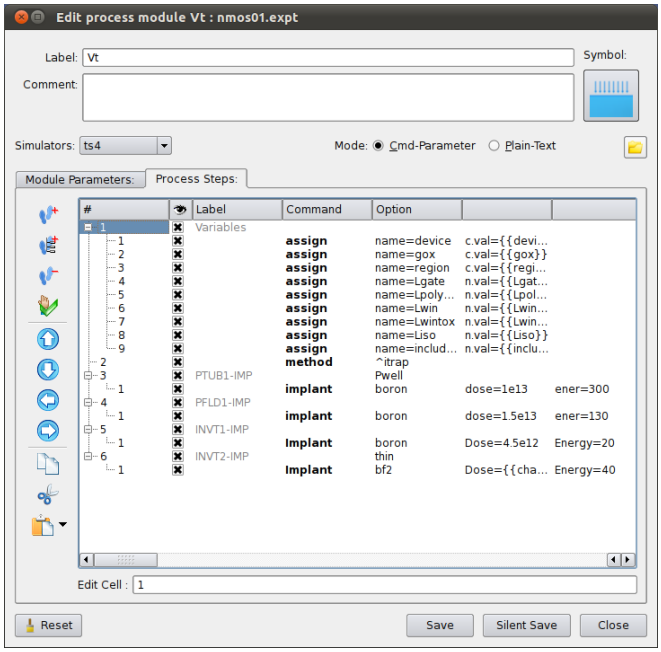

Figure 2.12 Editing a process module

Steps in Process Module

The core component of a process module is the sequence of steps. In general each step corresponds to a command for the process simulator. Every step has a

command and several options. An additional label can be added for description and comments.

Process steps can be nested. As shown in **Figure 2.12**, **p. 12**,the step *Variables* is a group of 9 sub-steps. The higher level step only serves the purpose of grouping, and the steps at the lowest level are used to generate commands for simulation.

Defining Parameters

We can define some parameters in a process module, as shown in **Figure 2.13**, **p. 14**. Each parameter must has a label and a default value. The parameters defined in the module is only visible within this module, effectively a local variable.

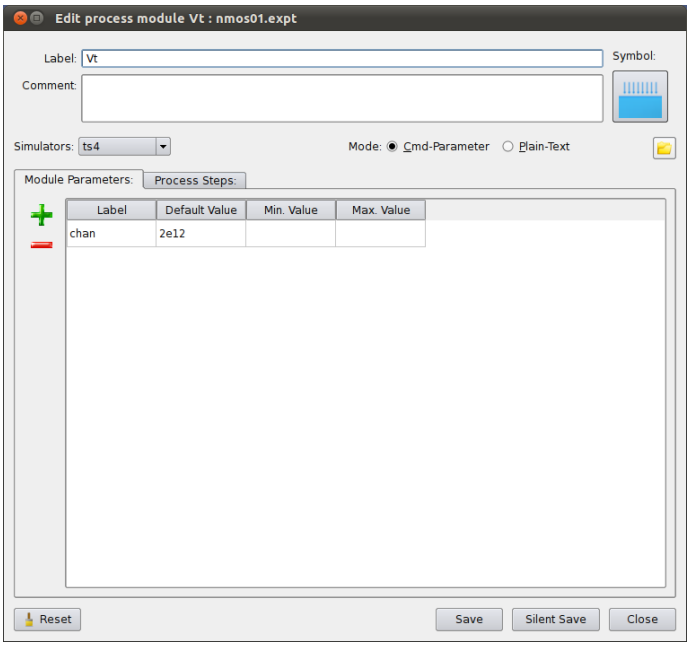

Figure 2.13 Defining Parameters in a Process Module

Using Parameters and Expressions

One noticed in Figure 2.12, p. 12 that in the last command we used the parameter `chan` with the syntax `{{...}}`.

```
Implant bf2 Dose={{chan}} Energy=40 tilt=-7 rota=45 ...
```

The curly braces indicate that the content is an expression in the Python programming language. In the simplest form, the above expression is the parameter we just defined. Before VisualFab invokes the simulators, it evaluates all the expressions and substitutes the result back, so that the final command becomes

```
Implant bf2 Dose=1e12 Energy=40 tilt=-7 rota=45 ...
```

Expressions can contain mathematics formulae as well. For example, the followings are valid expressions.

```
Implant bf2 Dose={{1e12 * 2}} Energy=40 tilt=-7 rota=45 ...
Implant bf2 Dose={{eval(chan) * 1.05}} Energy=40 tilt=-7 rota=45
...
Implant bf2 Dose={{chan}}*1.05 Energy=40 tilt=-7 rota=45 ...
```

After evaluation and substitution, the corresponding final commands are

```
Implant bf2 Dose=2e12 Energy=40 tilt=-7 rota=45 ...
Implant bf2 Dose=1.05e12 Energy=40 tilt=-7 rota=45 ...
Implant bf2 Dose=1e12*1.05 Energy=40 tilt=-7 rota=45 ...
```

Apart from the module parameters defined by the user, there are two other types of parameters that can be used in expressions. The first is the user-defined global parameters, which will be discussed in the next section. The other type is the variables automatically generated by VisualFab. Some of the commonly used variables are listed in Table 2.1, p. 15.

Variable	Description
`cur_dir`	Path to the directory of the current wafer state.
`parent_dir`	Path to the directory of the parent wafer state.
`expt_dir`	Path to the directory of the experiment.

Table 2.1 Commonly used variables.

Variable	Description
`script_file`	Script file of the current module.
`output_file`	Output file of the current module.

Table 2.2 Commonly used variables.

Experiment Splits

The central function of VisualFab is to handle experiments with splits. These functions are accessible through the tool-bar in the run table window, as shown in Figure 2.14, p. 17.

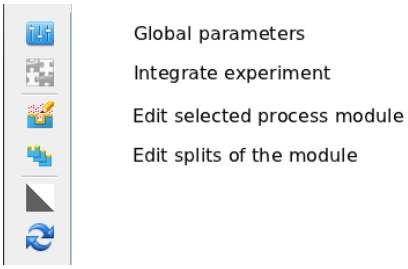

Figure 2.14 Tool-bar of the run table.

In the following sections, we shall look at the procedures of defining global parameters and making splits.

Defining Global Parameters

We can define parameters that are visible in all process modules, i.e. global variables, as shown in Figure 2.15, p. 17. Similar to process module parameters, the global parameters requires a label and default value, and can be used in expressions.

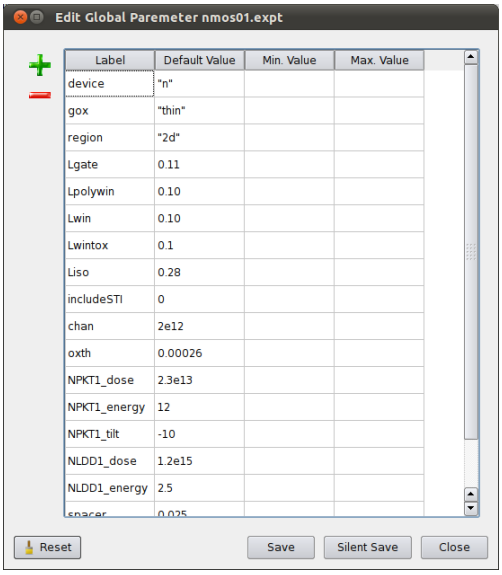

Figure 2.15 Defining global parameters.

Making Splits

In the run table window (**Figure 2.16**, **p. 18**), we see that the experiment is split at the *Nsd* module, on the `NPKT1_dose` parameter. To view and edit the split, we select the row of the `NPKT1_dose` parameter, and click the Edit Splits button.

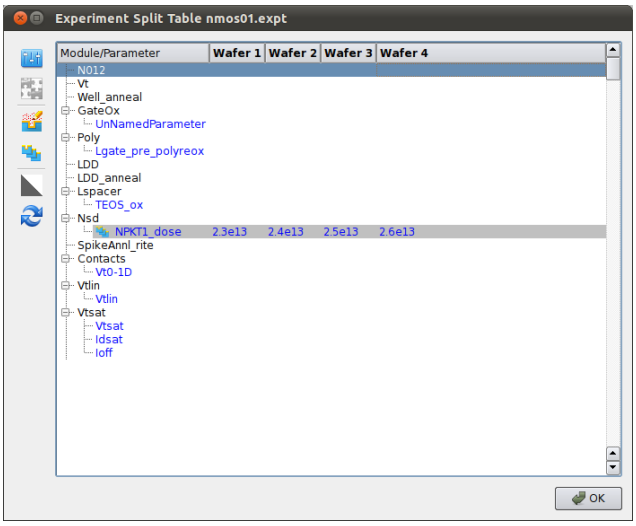

Figure 2.16 Run table of the NMOS experiment.

The splits editing dialog shows up as in **Figure 2.17**, **p. 18**. The four splits are shown in columns, and the splinted parameters in rows. Each split can be labeled, though the labels are not currently used.

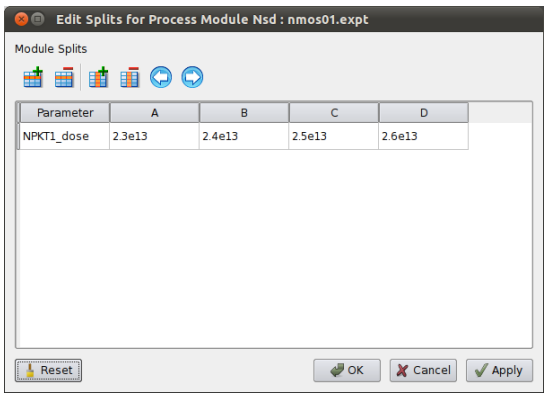

Figure 2.17 Editing splits of a process module

If the experiment topology is changed, e.g. by adding new splits, the experiment needs to be re-integrated with the Integrate button.

CHAPTER 3

Components of VisualFab

Process Modules

The process flow consists of several process modules. To view and edit a process module, one can double-click it in the process flow view. The process module dialog (**Figure 3.1**, **p. 19**) shows up with the following components:

1. Label. Each process module must have a distinctive name, Vt, Well_anneal, GateOx, etc. are some typical examples. Space and special characters are not allowed in label name.
2. Comment. Some textual description.
3. Symbol. A graphic symbol for the module.
4. Simulator. Select the process simulation tool that will be used to simulate this module.
5. Process Steps. The commands that will be passed to the tool for the simulation of his module.
6. Module Parameters. Some variables to be used in this module only. One can create split on module parameters.

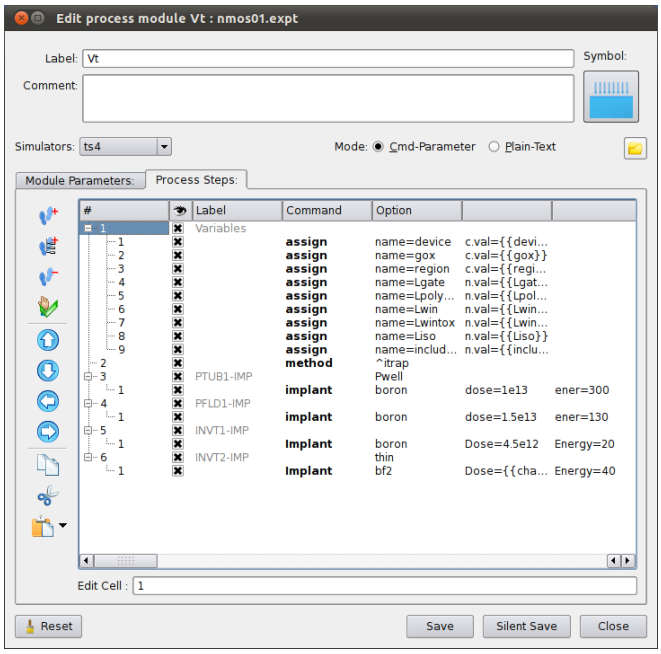

Figure 3.1 Editing a process module

Steps in Process Module

The core component of a process module is the sequence of steps. In general each step corresponds to a command for the process simulator. Every step has a command and several options. An additional label can be added for description and comments.

Process steps can be nested. As shown in Figure 3.1, p. 19,the step *Variables* is a group of 9 sub-steps. The higher level step only serves the purpose of grouping, and the steps at the lowest level are used to generate commands for simulation.

Figure 3.2, p. 20 shows the toolbar for managing the process steps in the process module. These tool buttons become active when one or more steps are selected. Process steps are selected by clicking in the first column (clicking in other columns would mean selecting a single cell).

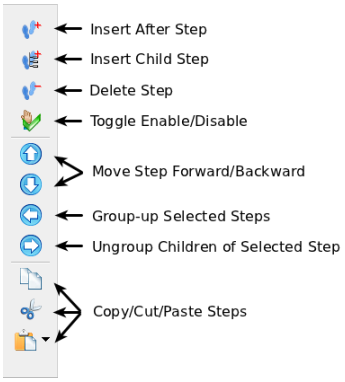

Figure 3.2 Toolbar of process module edit dialog.

Special Commands

Each step has a command name. Besides the commands defined by the simulator tools, there are a few special commands defined in VisualFab. One can click in the command cell, and activate the drop-down list for these special commands, as shown in Figure 3.3, p. 21.

- `SOURCE` command. Include the content of an external text file to the current process module. Takes one option, file to be included.
- `IF/ELSE` command. Evaluate the expression that follows the `IF` command, if the result is true, use the child-steps under the `IF` command, otherwise use the child-steps under the `ELSE` command.

When one of the special commands is selected, the cell will be displayed with green font color. On the other hand, if one manually keys in an IF commands, i.e. *not* using the drop-down list, the command is displayed in normal font color, and is treated as a normal command understood by the simulator program.

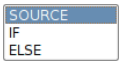

Figure 3.3 Drop-down list of special commands.

Command Options

One can edit the options of each command by clicking at the corresponding cell. A context menu (**Figure 3.4, p. 21**) is available for editing the options.

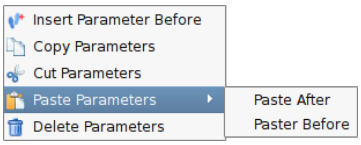

Figure 3.4 Context Menu

Save

One has the choice of Save and Silent Save when he wishes to save the modifications to a process module. This is related to the signature checking and dependency checking rules in VisualFab. When one choose to Save a module, and if the changes to it are essential ones (changes to the comments is non-essential), the signature of the process module is changed, and all wafer states that are derived from the module becomes invalid and requires to be simulated again.

On the other hand, if one chooses Silent Save, the signature of the process module is not updated, and wafer states not re-run.

Module Parameters

[[TODO]] Elaborate further from “**Defining Parameters**”, **p. 14**.

Network Machines

VisualFab can submit simulation jobs to remote computers. To manage and monitor the status of the remote machines, one can click in the main toolbar Workspace Manager, and the list of machines that are currently in use is shown in the dialog shown in **Figure 3.6, p. 23**.

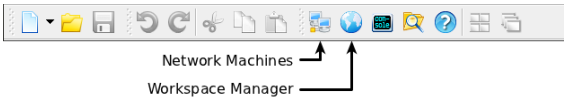

Figure 3.5 The VisualFab Main Tool-bar

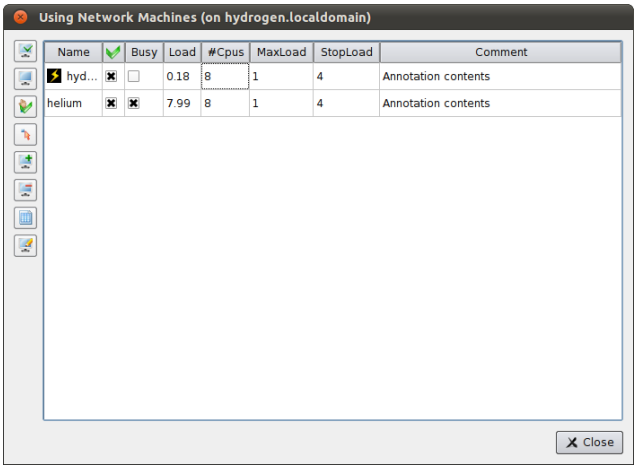

Figure 3.6 List of machines under active use.

The status of each machine is displayed in the table, and the columns are, from left to right,

- Machine name. It's the hostname configured by the system administrator and automatically detected when user search for machines. Alternatively, a user can manually set the machine name.
- Enabled. VisualFab will *only* submit to enabled machines, which has this column checked.
- Busy. If the machine's `Load` exceeds `StopLoad`, the machine is marked as busy, and VisualFab temporarily stops submitting job to this machine.
- Load. The machine's 1-minute average load factor. Its value is usually between 0 and the number of processors of the machine.
- MaxLoad. Not currently used.
- StopLoad. The load threshold above which VisualFab stops submitting jobs to the machine.
- Comment.

One can manage the machines, and view the jobs running on them with the toolbar shown in Figure 3.7, p. 24 or the context-menu in the machine list.

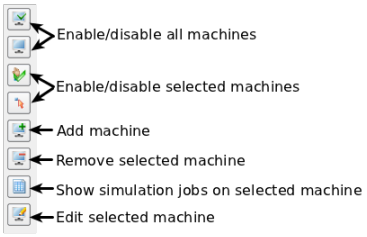

Figure 3.7 Toolbar of the network machine list dialog.

List of Jobs

When one double-clicks a machine in the list, or click Show jobs in the toolbar, the dialog shown in Figure 3.8, p. 24 appears, with the list of jobs running on the machine displayed in the dialog.

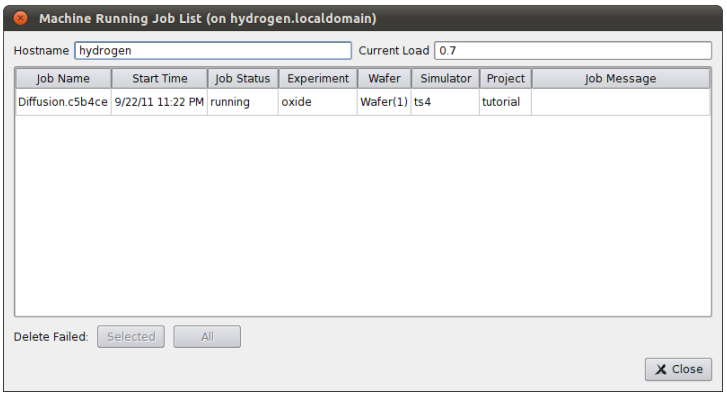

Figure 3.8 List of jobs running on the machine.

Edit Machines

One can edit the selected machine with the Edit machine button. The dialog for editing the properties of the machine is shown in Figure 3.9, p. 24. In the `Address` field, one can enter either IP address or hostname of the machine.

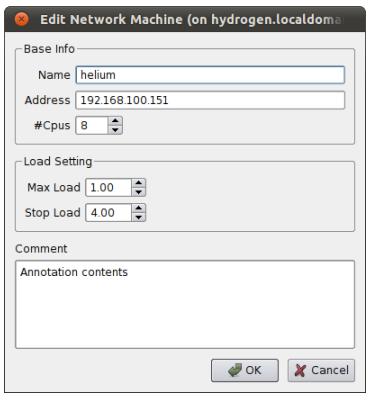

Figure 3.9 List of known machines.

Adding Machines

One can add machines to the list by clicking the Add Machine button. The dialog shown in Figure 3.10, p. 25 lists all available machines known to VisualFab. One can select some machines, a click the Add to Using List button, so VisualFab will start using them.

If the machine is not yet known to VisualFab, one can either add machine manually with the Add Machine button, or the Search Machines button.

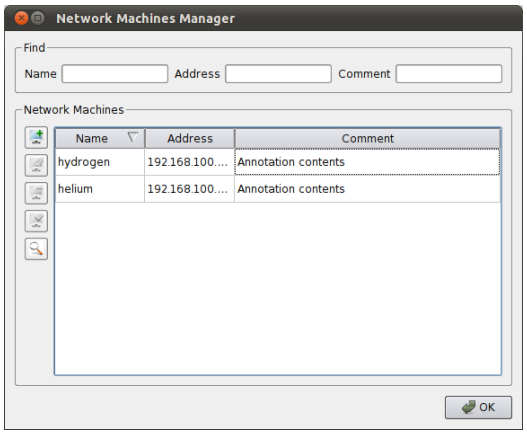

Figure 3.10 List of known machines.

In the search machine dialog (Figure 3.11, p. 25), one can instruct VisualFab to search for machines in the local area network.

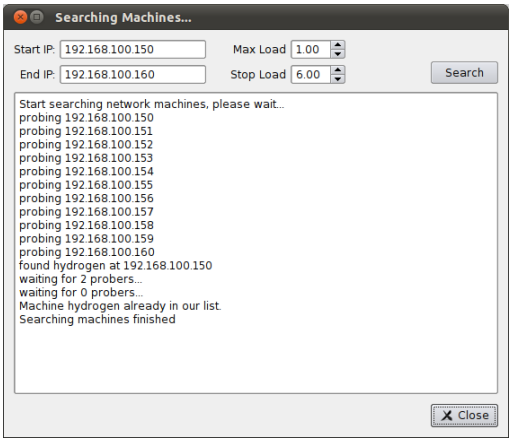

Figure 3.11 List of machines under active use.

Currently VisualFab only recognize machines that accepts RSH login without requiring a password, and detects the number of processors. The user can specify the Stop-load of the machine.

Workspace and Projects

The workspace manager shown in **Figure 3.12**, **p. 26** is usually the first window to open when one uses VisualFab. In the workspace manager, one can browse the projects and experiments in the workspace, open an experiment, create new projects or experiments, and copy/move experiments around.

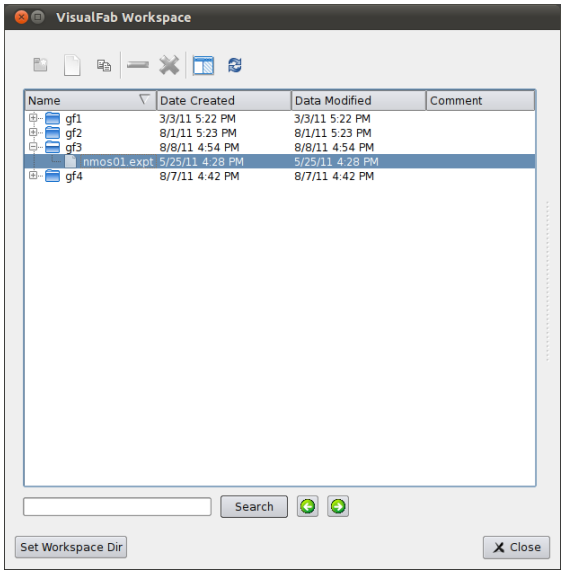

Figure 3.12 Workspace Manager

In the tree-like view, experiments are displayed as files, while projects are displayed as folders. Projects can contain nested sub-projects. Projects and experiments can be sorted by name, creation date or modification date. Experiments can have comments as well.

The toolbar for the workspace manager is shown in **Figure 3.13**, **p. 26**. Most of the functions are also available in the context menu when one selects a project or an experiment, as shown in **Figure 3.14**, **p. 27**.

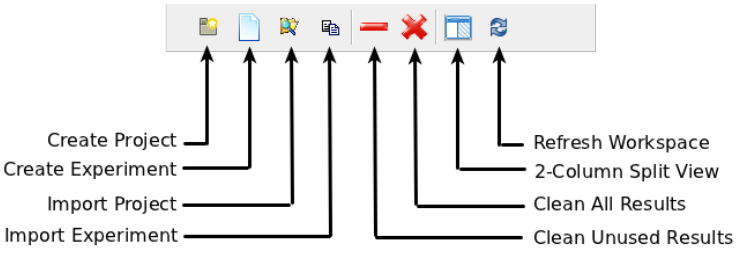

Figure 3.13 Toolbar of workspace manager

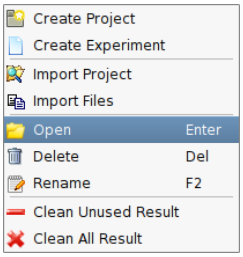

Figure 3.14 Context menu for projects and experiments in the workspace

Searching for Experiment

If one wants to look for an experiment, one can type its partial name in the search box (1) shown in **Figure 3.15, p. 27**, and click Search (2). The matches will be highlighted in the workspace (4), and one can skim through the matches with the previous (2) and next (3) buttons.

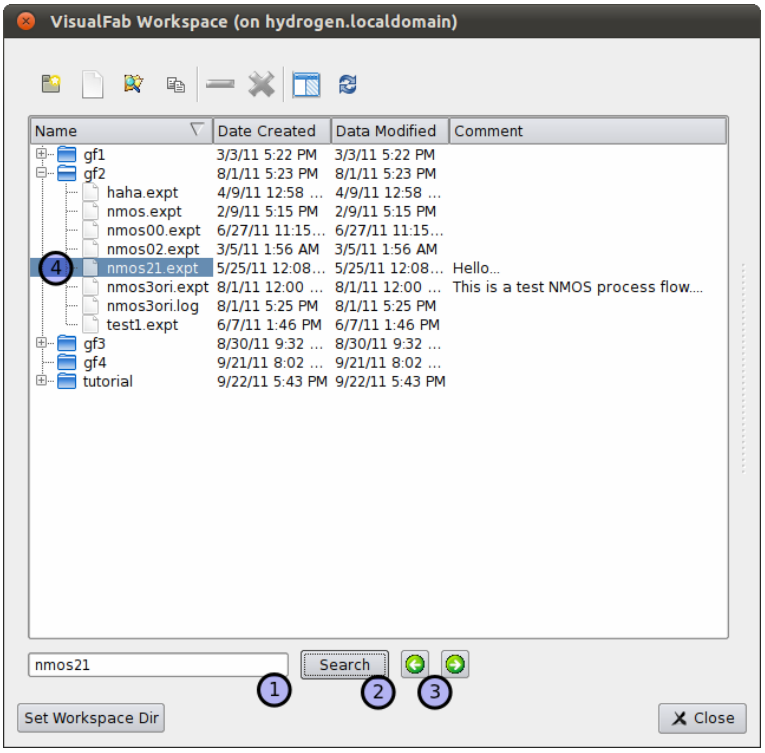

Figure 3.15 Search for experiments.

Copying Experiment

The One/two-column view button in the tool-bar allows the user to split the workspace view, as shown in **Figure 3.16**, **p. 28**. Copying and moving are easier in the two-column view. One can drag an experiment from one project, and drop it to another. A menu will then pop-up (**Figure 3.17**, **p. 28**), offering three options:

- Shallow copy. Only the experiment setup is copied, while the simulated wafer states are not.
- Deep copy. The simulated wafer states are copied along with the experiment setup.
- Move. The experiment setup is moved to the new project.

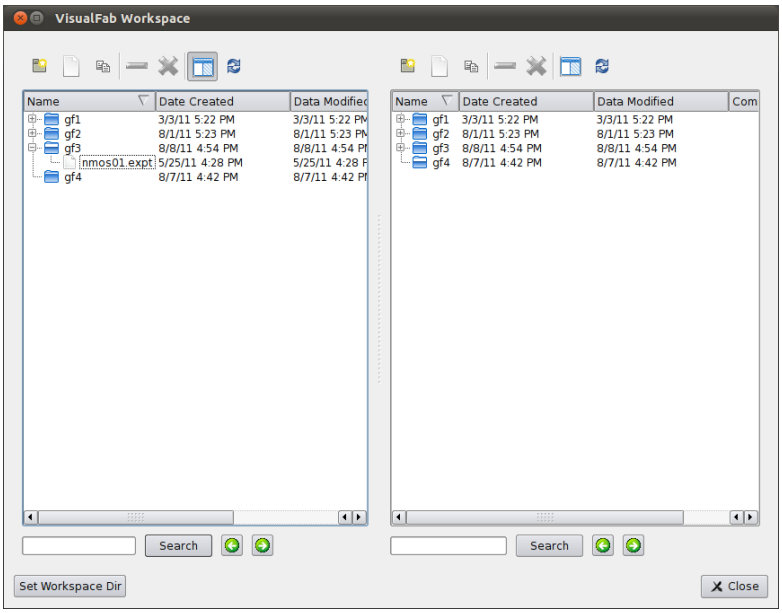

Figure 3.16 Workspace Manager in split view.

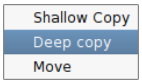

Figure 3.17 Options to copy, deep-copy, or move experiments and projects.

Importing Experiments and Projects

It is possible to import experiments from `.expt` or `.vfl` files outside the workspace, with the Import File button. If one imports a `.vfl` file (TWB format), VisualFab attempts to convert it to the .expt format, as shown in **Figure 3.18**, **p. 29**.

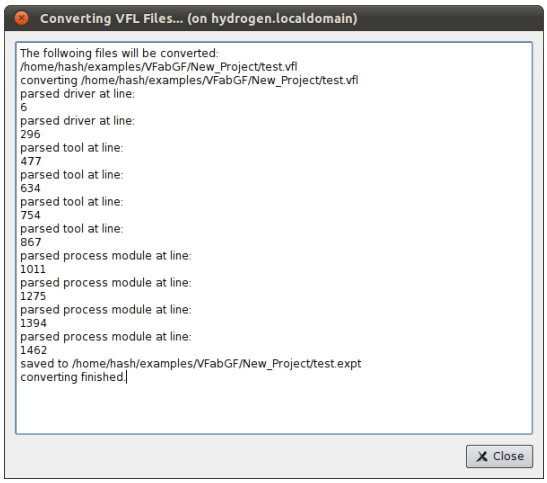

Figure 3.18 Converting .vfl project

One can also import an entire project from a directory. In this case, the project is not copied to the workspace. Instead, only a symbolic link is created.

The experiments in the imported projects are read-only, so one can open them but not saving changes to them. One can copy an experiment from imported project to a native project in the workspace, after which the experiment become editable.

Simulators and Inspectors

VisualFab relies on external tools for simulation, visualization and data processing. This chapter describes the simulator and inspector tools.

Managing Simulators

Simulators are the programs that performs process or device simulation, and each process module is associated to a simulator. Simulators typically import a device structure from a wafer state, execute the process steps specified in the process module, and output a new wafer state.

Simulators are listed in the toolbox in the left-hand side of the experiment window, as shown in **Figure 3.19**, **p. 30**. One can add/edit/delete simulators with the tool button or the context menu.

Figure 3.19 List of simulators in the toolbox.

Double-clicking a simulator invokes the editing of a simulator as well, and the dialog is shown in **Figure 3.20**, **p. 31**.

Each simulator is identified by a label. One may assign an icon to represent the simulator, and optionally add some text description as comment. The simulators are described with properties in five main sections: machine, unix, load, save and options.

Before running simulator for a process module, VisualFab creates a temporary directory for the target wafer state, in the project directory (where the `.expt` file is located).

It then creates an input command file (e.g. `commands.inp`) in the syntax of the simulator tool in this directory. The chunk of the input commands come from the process module, while the load/save sections define the pre- and post-processing steps.

VisualFab also creates a shell script (e.g. `run.sh`) in the directory, using the definitions in the unix section.

VisualFab then selects a machine from the list in the machine section to run the simulation . The choice is made according to the availability and loading of machines. Finally, VisualFab logon to the selected remote machine, enters the temporary directory for the target wafer state, and runs the shell script to start the simulation.

Machine

As shown in **Figure 3.20**, **p. 31**, this section contains the list of machines on which the simulator software is installed.

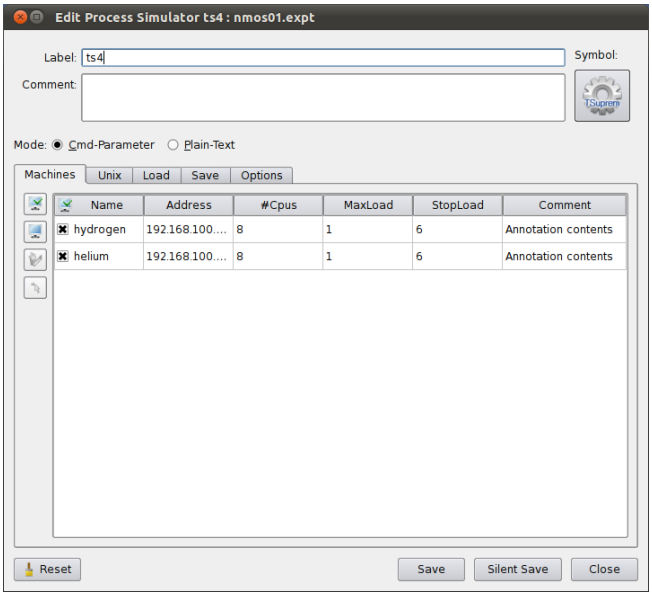

Figure 3.20 Machine section in the simulator dialog

Unix

A simulator is usually a unix program. This section defines the command and arguments to run this simulator program. As shown in **Figure 3.21**, **p. 32**, one can write a shell script, which typically contains environment exports, path to the program and arguments to the program.

Variable substitution is available in the shell script definition. The expression `{{cur_dir}}` will be expanded to the path to the directory of the current wafer

state (more precisely, the temporary directory of it). Since simulator program will be started in that directory, the use of the expression is optional here, and one could simply write `commands.inp`. For a list of variables, please see Table 2.1, p. 15.

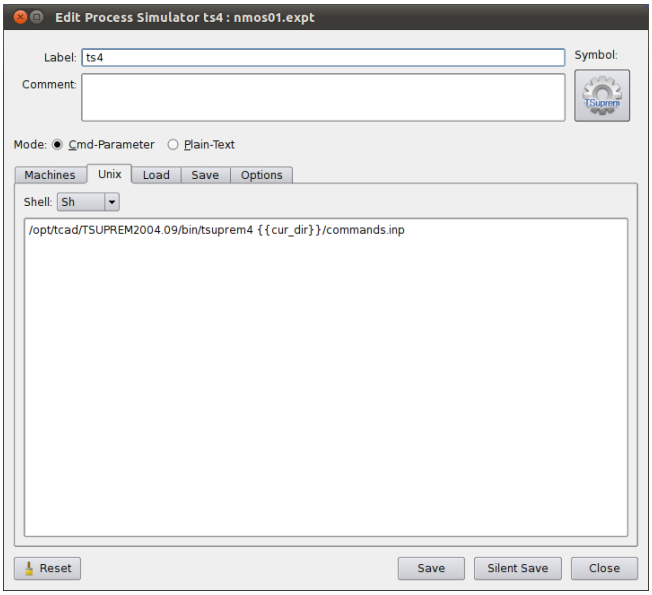

Figure 3.21 Unix section in the simulator dialog

Load and Save

Each simulation job is mainly defined in a process module, which consists of a series of steps (commands). Some commands are common for every simulation job. For example, at the beginning of each simulation, one wants to load the device structure produced by the previous process module; at the end of the simulation, one wants to save the device structure to be used by the next process module.

The load and save sections, shown respectively in Figure 3.22, p. 33 and Figure 3.23, p. 33, allow users to specify the common steps for all process modules using this simulator. Commands defined in the load section appear at the beginning of the `commands.inp` file, while the commands in the save section appear at the end.

Options

In the options section, shown in Figure 3.24, p. 34, user can tune some behavior VisualFab runs simulation.

- Script file. The file to which process step commands are saved to.
- Comment character. The character(s) that starts a comment line in the script file.
- Output file. The file which the simulation creates when it finishes. VisualFab checks this file to determine if the simulation completes successfully. Typically, in the save section of the simulator, one saves the device structure to the output file..

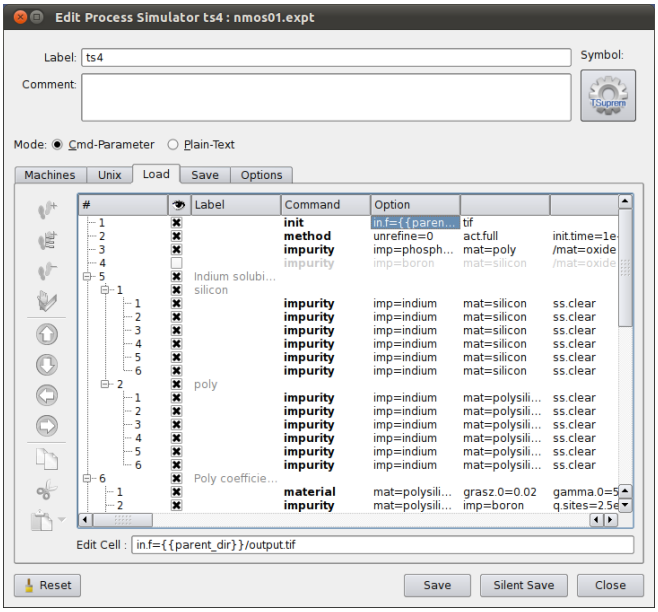

Figure 3.22 Load section in the simulator dialog

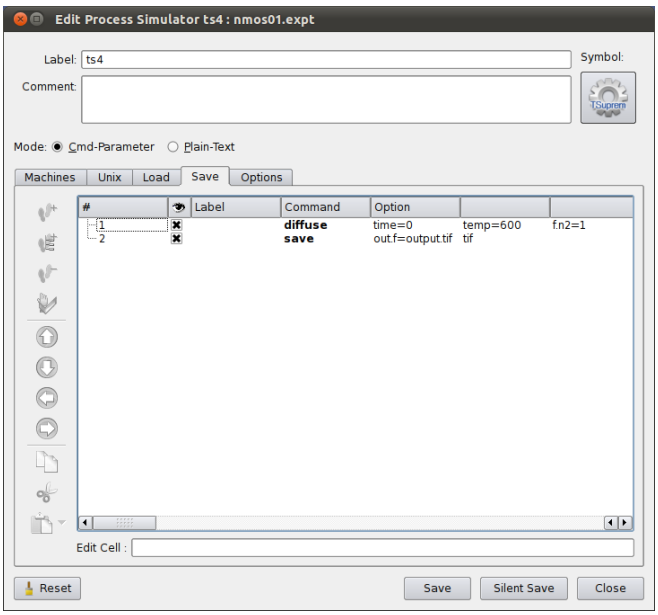

Figure 3.23 Save section of the simulator dialog

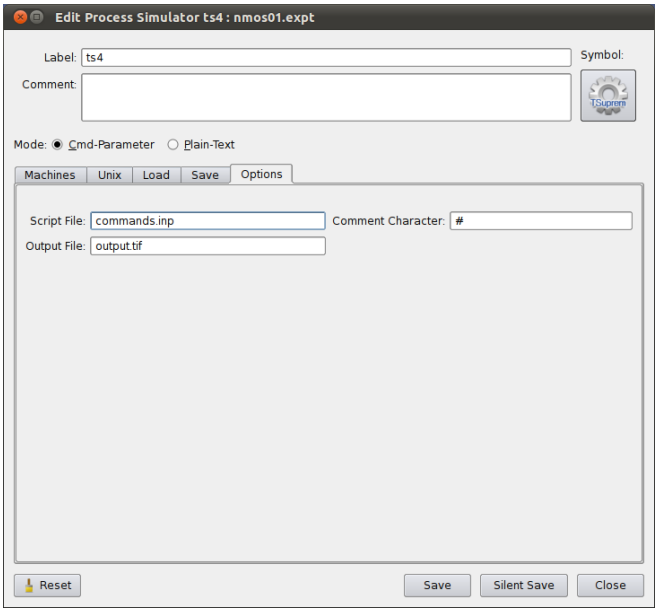

Figure 3.24 Option section of the simulator dialog

Managing Inspectors

Inspectors are programs that help users examine one or more wafer states, i.e. to visualizes a device structure or post-process simulation results. Inspectors are listed in the toolbox in the left-hand side of the experiment window, as shown in **Figure 3.25**, **p. 35**. One can add/edit/delete simulators with the tool button or the context menu.

Figure 3.25 List of inspectors in the toolbox.

Double-clicking an inspector invokes the editing of an inspector as well, and the dialog is shown in **Figure 3.26**, **p. 36**. The label, comment and icon fields are similar to those in simulators. The inspectors are described with properties in four main sections: unix, begin, command, end and options. Inspectors are run on the local machine, and therefore do not have a machine section.

When instructed to inspect a wafer state, VisualFab first creates a temporary directory under the `Inspector/` under the project directory.

It then creates an input command file (e.g. `commands.inp`) in the syntax of the simulator tool in this directory. The input commands come from the begin/command/end sections. VisualFab also creates a shell script (e.g. run.sh) in the directory, using the definitions in the unix section.

For each wafer state instructed to examine, VisualFab finds the output file, and creates a link to it in the temporary directory. Finally, VisualFab enters the temporary directory , and runs the shell script (`run.sh`) to start the inspector.

Unix

An inspector is usually a unix program. This section defines the command and arguments to run this simulator program. As shown in **Figure 3.26, p. 36**, one can write a shell script, which typically contains environment exports, path to the program and arguments to the program.

Variable substitution is available in the shell script definition. For a list of variables, please see **Table 2.1, p. 15**.

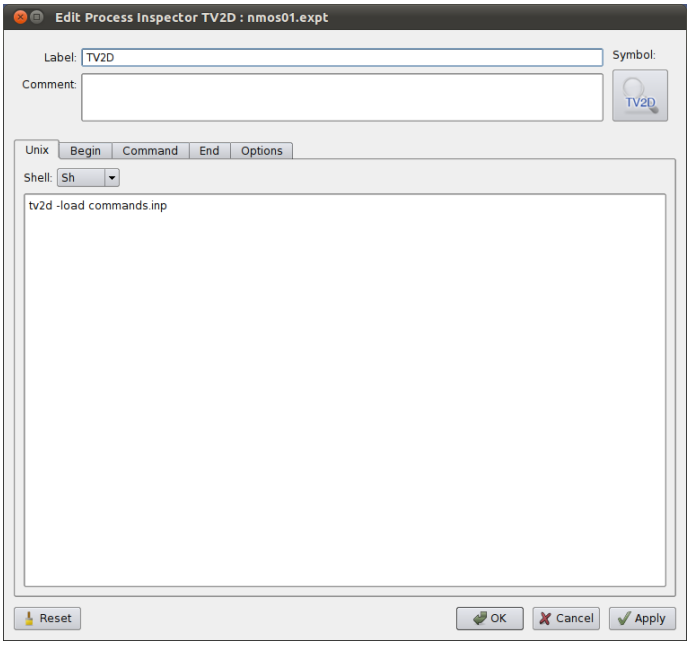

Figure 3.26 Unix section of the inspector dialog.

Options

In the options section, shown in **Figure 3.27, p. 37**, user can tune some behavior VisualFab runs simulation.

- Script file. The file to which process step commands are saved to.
- Comment character. The character(s) that starts a comment line in the script file.
- Output file pattern. The simulation output file (or files) which the inspector is to examine, wildcasts like `*.tif` can be used in this field to match multiple files.

For each wafer state is instructed to examine, VisualFab finds the output files that match the output file pattern from wafer state's directory. Matched files from all wafer states will be examined by the inspector.

Begin/Command/End

Many visualization and post-processing programs takes a command script, and VisualFab is responsible for creating the script.

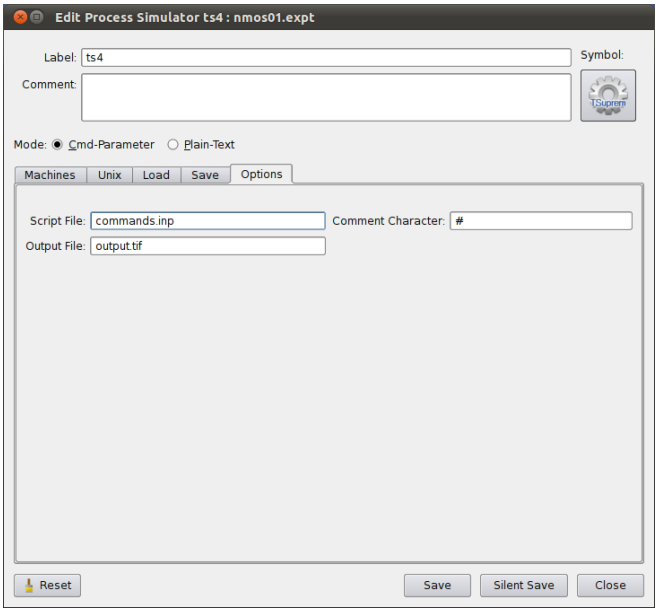

Figure 3.27 Option section of the simulator dialog

The commands in the begin section will be added to the script first. Then the commands in the command section will be added for each matched file to examine. Commands in the end section will be added lastly.

As explained a bit earlier, a link is created in the temporary directory for each matched file. The variable `matched_file` contains name of the link, which is only applicable in the command section.

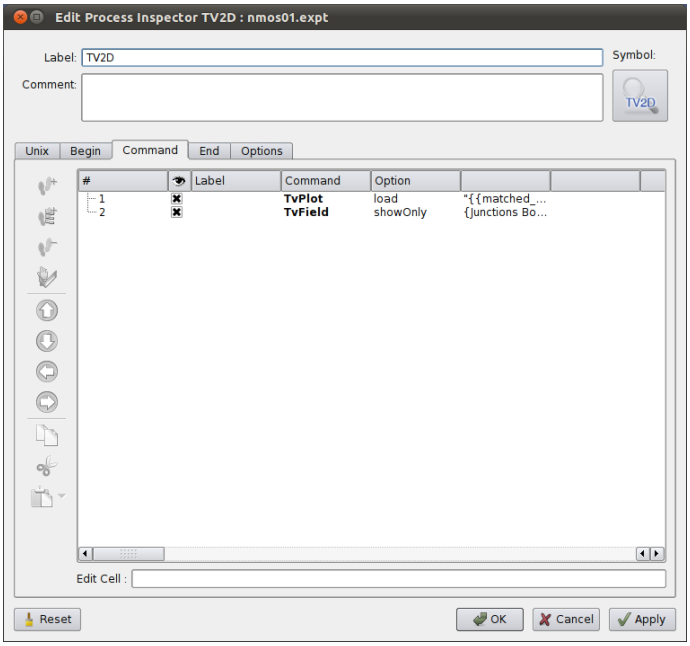

Figure 3.28 Command section of the simulator dialog

Using Inspectors

To examine some wafer states, one can select them in the experiment wafer state view, drag them and drop to an inspector. The inspector program will be started, and each wafer state will be loaded and displayed.

Alternatively, one can right-click on the wafer state, and select Begin Receive States ▷ in the context menu in **Figure 3.29**, **p. 38**. One can drop wafer states to the inspector, but the inspector is not started immediately. After all wafer states have been dropped to it, one can select End Receive States ▷, and only then the inspector will be started. With this mechanism, it is also possible to drag wafer states from several experiments, and open them in the same inspector.

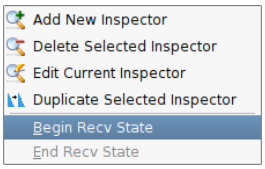

Figure 3.29 Context menu of inspector.

Process Flow

Global Parameters

[[TODO]] Elaborate further on “**Defining Global Parameters**”, **p. 17**.

Experiment Splits

[[TODO]] Elaborate further on “**Making Splits**”, **p. 18**.

Utilities

File Explorer

Figure 3.30, **p. 41** shows the file explorer window. It can be activated in the main toolbar. One can navigate the file system, and open documents with it. One can also create, rename and delete files and directories here.

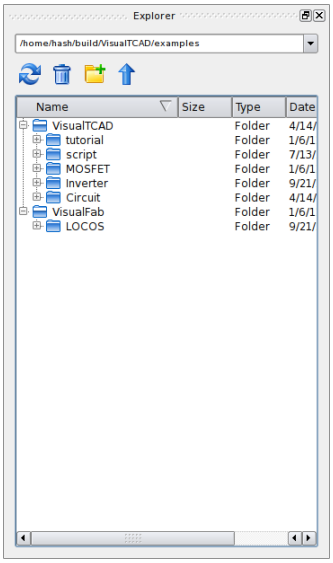

Figure 3.30 File Explorer

Setting Profile

The behavior of VisualFab is affected by many setting parameters. One can edit the settings from the menu Edit ▷ Setting Manager. As shown in **Figure 3.31**, **p. 42**, one can have several setting profiles, and choose one active profile from the list.

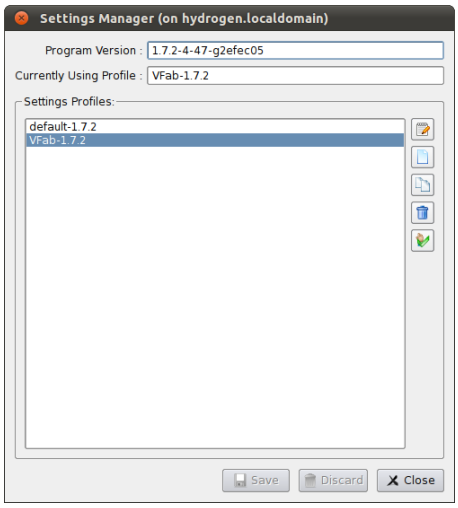

Figure 3.31 Select the setting profile.

One can edit parameters in a setting profile in the dialog **Figure 3.32**, **p. 42**.

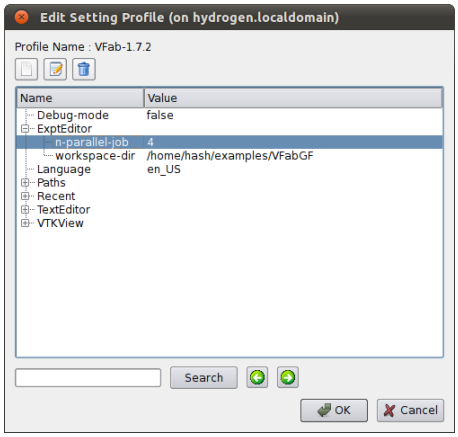

Figure 3.32 Editing parameters in a setting profile.

Log Console

One can monitor the activities of VisualFab in the log console, by clicking the Console button in the main toolbar.

Messages can be filtered by the types (2): Error, Warning, Information, or Debug; messages can also be filtered by the source of the messages (3), i.e. from which module the message is emitted.

One can export messages a to text file (4), or to clean up the message from the console window.

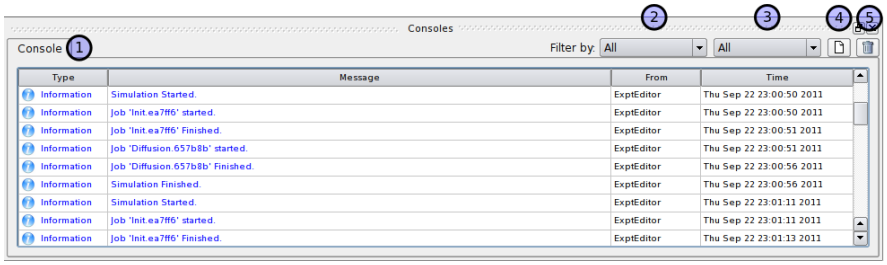

Figure 3.33 Log console window.