Sentaurus™ Calibration Workbench User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

Contents

About This Guide

This user guide describes the Synopsys® Sentaurus™ Calibration Workbench tool.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Bold text	Identifies a selectable icon, button, menu, or tab. It also indicates the name of a field or an option.
`Courier font`	Identifies text that is displayed on the screen or that the user must type. It identifies the names of files, directories, paths, parameters, keywords, and variables.
Italicized text	Used for emphasis, the titles of books and journals, and non-English words. It also identifies components of an equation or a formula, a placeholder, or an identifier.
Menu > Command	Indicates a menu command, for example, **File > New** (from the **File** menu, choose **New**).

Customer Support

Customer support is available through the Synopsys SolvNetPlus support site and by contacting the Synopsys support center.

Accessing SolvNetPlus

The SolvNetPlus support site includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The site also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Support Center.

To access the SolvNetPlus site:

1. Go to https://solvnetplus.synopsys.com.
2. Enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register.)

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys Global Support Centers site on www.synopsys.com. There you can find email addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNetPlus site or the Synopsys Global Support Centers site and open a case (Synopsys user name and password required).

Contacting Your Local TCAD Support Team Directly

Send an email message to:

- support-tcad-us@synopsys.com from within North America and South America
- support-tcad-eu@synopsys.com from within Europe
- support-tcad-ap@synopsys.com from within Asia Pacific (China, Taiwan, Singapore, Malaysia, India, Australia)
- support-tcad-kr@synopsys.com from Korea
- support-tcad-jp@synopsys.com from Japan

CHAPTER 1

Introduction to Sentaurus Calibration Workbench

This chapter introduces the functionality of Sentaurus Calibration Workbench.

Overview of Functionality

Sentaurus Calibration Workbench is a batch tool designed to perform efficient calibrations for TCAD simulation tools. It provides the following functionality:

- *Evaluation* allows you to run the nominal or current parameter set for given target specifications, for validation of the simulation setup and extraction of the current accuracy.
- *Calibration* allows you to find an optimal parameter set for given target specifications. Multiple calibration targets can be combined, with weights specifying their relative importance. Several algorithms are available to solve calibration problems.
- *Sensitivity analysis* determines how small changes to a given parameter affect simulation responses and determines primary parameters.
- *Search* allows you to run different designs-of-experiments (DoE) to explore the parameter space.
- *Machine learning* allows you to train models and use them to explore the parameter space to find an optimal parameter set for given target specifications.
- *Predefined or user-defined calibration strategies* including several calibration tasks can be applied to ensure efficient and reproducible calibration processes.

Sentaurus Calibration Workbench provides a Python environment where you can define and use custom algorithms that address specific tasks. All the components of Sentaurus Calibration Workbench are available in this environment and can be combined to develop a calibration strategy. For example, you can combine a sensitivity analysis with a calibration algorithm: through the sensitivity analysis, the primary parameters for calibration are found, and then only these are calibrated.

Although Sentaurus Calibration Workbench is a standalone tool, it interacts with the batch tools of Sentaurus Workbench for setting up and running simulations (Sentaurus Workbench experiments). It can take advantage of the job scheduler of Sentaurus Workbench to speed up simulations by using distributed heterogeneous computing resources.

Sentaurus Calibration Workbench features are available in different tiers, as follows:

Feature	Base	Elite	Apex
Sentaurus Workbench integration: automatic project creation, job submission, and data exchange	X	X	X
Sensitivity analysis and calibration strategies based on traditional techniques; all functions that require machine learning are excluded	X	X	X
Machine-learning functions, including calibration strategies based on machine learning; machine-learning API is excluded		X	X
Machine-learning API that allows you to include custom-defined machine-learning models			X

Basic Concepts and Terminology

Common terms relevant to understanding Sentaurus Calibration Workbench are:

- A *parameter* is a scalar variable, usually defined in a Sentaurus Workbench project, that modifies the simulation flow. It has a finite value that is defined inside a certain domain. There are different parameter categories:
  - *Calibration parameters* are calibrated during the calibration process.
  - *Condition parameters* are not calibrated but define split conditions or states of experiments (for example, conditions in which the target data was obtained experimentally). Their values must not be modified while a calibration is running. However, depending on their purpose, you can vary *state parameters* so Sentaurus Calibration Workbench can calculate specific targets. For example, you can change the drain voltage to calculate separate I_d–V_g curves.
  - *Primary parameters* are defined as parameters with high sensitivity.
- A *response* is a variable or simulation output that describes, for example, device or circuit behavior. A response could be the error of the threshold voltage for the target device with respect to the required value.
- A *target* is a response to be matched against a reference by calibration. The reference could be based on measurements or calculations.
- *An experiment or a parameter set* is a tuple with one value for each calibration parameter and condition parameter that defines a particular instance of the simulation flow.
- *A family of simulations* is a set of split simulations that have the same set of calibration parameters but a different set of condition parameters.
- An *evaluation* is defined as all the simulations required to compute a given family of simulations.

- An *iteration* is an evaluation for a given calibration parameter set as part of a calibration task.
- A *task* is a specific action that Sentaurus Calibration Workbench performs. For example, a task could be a sensitivity analysis, a gradient-based local calibration of a set of parameters, or a machine-learning model generation. Tasks can consist of *subtasks*.
- A *calibration strategy* or *calibration flow* is a sequence of tasks used to obtain information about the relationship between the parameters and responses under consideration.
- A *step* or *calibration step* is a numbered task of a calibration strategy.
- A *trial* or *calibration trial* is the execution of a calibration strategy. If there is variability in the simulator results, then calibration trials can give different results as well.
- A *parent project* is a Sentaurus Workbench project that contains all of the tool steps to be included in the calibration, but not Sentaurus Calibration Workbench itself. Moreover, it includes the variable extractions that are linked to calibration targets. In general, it also includes all calibration parameters as Sentaurus Workbench parameters, with nominal values.
- A *child project* is a Sentaurus Workbench project that Sentaurus Calibration Workbench generates. A child project is based on a parent project and includes all experiments of the calibration.

Syntax Conventions

For details about the syntax conventions used for the Python commands in this user guide, see Syntax Conventions on page 89.

Running Sentaurus Calibration Workbench

You can run Sentaurus Calibration Workbench in different ways:

- From within Sentaurus Workbench by introducing Sentaurus Calibration Workbench as a tool in the simulation flow of a Sentaurus Workbench project.

  The Sentaurus Calibration Workbench Python command file or script can be specified for the Sentaurus Calibration Workbench tool by right-clicking its tool icon and choosing **Edit Input > SCW Script**. The Python script contains the description of the calibration process to be performed.

- From the command line by using the command:

  ```
  scw <options> <input_file>
  ```

  where `<options>` are command-line options (see Command-Line Options) and `<input_file>` is a Sentaurus Calibration Workbench input file (see Input File).

Command-Line Options

The following command-line options are available for Sentaurus Calibration Workbench:

```
-h[elp]              : Displays this help message
-v[ersion]           : Displays version information
-verbose             : Displays additional initialization and loading
                       information
-node "string"       : Launches calibration for given node
-prevnode "string"   : Specifies the previous node of the given node
```

Input File

The Sentaurus Calibration Workbench tool instance has one input file, which is a Python script describing the calibration setup. This script is described in Chapter 2 on page 5. You can access this file by right-clicking the tool icon and choosing **Edit Input > Commands**. Within a Sentaurus Workbench project, this file can contain Sentaurus Workbench preprocessor directives.

NOTE Child projects are not deleted by default when cleaning up nodes.

CHAPTER 2 Calibration Flows

This chapter explains how to set up calibration flows.

Structure of a Calibration Flow

The calibration flow or strategy of Sentaurus Calibration Workbench is defined in its input command file or script and is the main object that links all relevant elements. A calibration flow usually requires the evaluation of several combinations of different parameter sets and acts in accordance to different response values. These evaluations are performed through an external simulation process invoked and coordinated by Sentaurus Calibration Workbench.

Multiple tasks can be performed in one strategy. Strategies use the evaluations and information obtained from different tasks and responses, and organize them to achieve the required result. For example, if a sensitivity analysis task selects some calibration parameters, then a strategy can instruct subsequent tasks to use only those parameters.

If many calibration parameters should be calibrated, then a sequence of calibration tasks, each of them for only a subset of the calibration parameters, is often a recommended strategy.

In summary, a calibration flow or strategy is a sequence of tasks used to obtain information about calibrated parameters for a given set of targets. Sentaurus Calibration Workbench allows the development of user-defined calibration strategies for different calibration problems.

Calibration Modules

A calibration flow is based on an imported Sentaurus Calibration Workbench calibration module. Two types of calibration module are provided:

- A *basic* calibration module provides a basic application programming interface (API), which allows you to set up your own calibration solutions for any application. The available basic calibration modules are:
  - Sentaurus Calibration Workbench module, described in this chapter
  - Machine-learning module, described in Chapter 3, Sentaurus Calibration Workbench Machine-Learning Module
- An *application-specific* calibration module is designed for a specific application and has predefined calibration strategies developed by calibration experts. These expert calibration

strategies are contained in a function with a specific list of arguments and some prerequisites. They are included in application-specific calibration modules.

The application-specific calibration module is described in Chapter 6, Modules for Calibrating Topography Simulations.

The expert calibration strategy is described in HARC Etching Calibration on page 69.

Setting Up a Calibration Flow

First, you import a calibration module. For example:

```
from scw.apps.scwcal import SCWCal
```

This loads the basic Sentaurus Calibration Workbench module, described in this chapter.

Second, you instantiate a calibration object from the imported module class. For example:

```
cal = SCWCal()
```

Now, you can use this calibration object to manage the calibration flow.

For each calibration module, you then must define the calibration input (see Calibration Input on page 7) and the details for performing the calibration (see Performing Calibrations on page 16).

Some calibration modules also include predefined expert calibration strategies.

Finally, the calibration output can be further processed (see Calibration Output on page 24).

In general, the goal of calibration strategies is to minimize the root mean square (RMS) error of the target evaluations (see Calculating the Root Mean Square Error on page 25).

NOTE The Sentaurus Calibration Workbench input command file follows the syntax of Python scripts, which means the command file is case sensitive and indentation must be consistent. Comments start with the hash character (`#`) and can be appended to any line. The script is interpreted sequentially, so every variable must be defined before it is used. Apart from this, the order in which parameters, targets, and strategies are defined is arbitrary.

Calibration Input

The calibration input includes calibration parameters, targets, the simulation flow, and settings. The simulation flow is contained in a Sentaurus Workbench project and is called the *parent project*. The parent project as well as the settings are defined as options.

The calibration input is usually defined at the beginning of the calibration flow, but in principle, it can be defined at any position.

Moreover, parameters and targets can be included and excluded individually for each task of the calibration strategy. In addition, options can be set anytime and multiple times.

The following sections explain the parameters, targets, and options.

Parameters

Calibration parameters are represented by the `Parameter` class, which is imported automatically by:

```
from scw.core.parameter import Parameter
```

Calibration Parameters

You can add calibration parameters by using the following command:

```
p1 = Parameter(name=<s>, value=<f>, minimum=<f>, maximum=<f>, [transform=<o>])
```

A parameter is defined by a `name`, which must correspond to the name of a parameter as defined in the Sentaurus Workbench parent project (case sensitive) and the parameter in the simulator command file or parameter file. The Sentaurus Workbench parameter syntax (`@name@`) must be used for parameters in parameterized files.

NOTE If the parameter is not yet defined in the Sentaurus Workbench parent project, then it can be added implicitly (see Defining Extra Tool Parameters on page 15).

The initial value of the parameter is specified with `value`. The range of values that the parameter can take is specified by a lower value (`minimum`) and an upper value (`maximum`). You can optionally specify a transformation object for the parameter value (`transform`). By default, `transform=Linear`, but it can also be specified as `Log10`.

See Parameter on page 113.

Excluding and Including Parameters in Calibration Tasks

When parameters are defined, they should be assigned to a Python variable. To be included in subsequent calibration tasks, the parameter (object) variable should be included in a parameter (object) list:

```
parameters = [p1]
```

Here, `parameters` is the name of the list.

Multiple parameters can be included in a parameter list:

```
parameters = [p1, p2, p3]
```

Parameters that are not included in that list retain their value in the calibration task. Therefore, if a parameter should *not* be calibrated in a task, then it should not be included in the parameter list of the subsequent calibration tasks.

Attributes for Parameters

You can retrieve and set the attributes of a `Parameter` object. The `Parameter` class has the following attributes:

- `name`: Name of a parameter as defined in the Sentaurus Workbench project (case sensitive)
- `value`: Initial value of the parameter
- `minimum`: Minimum value that the parameter can accept
- `maximum`: Maximum value that the parameter can accept
- `transform`: Transformation function applied to the parameter value

For example, to retrieve the current value of the parameter `par`, use the variable:

```
par.value
```

To update the current value, use the assignment:

```
par.value = <f>
```

Targets

Calibration targets are represented by the `Target` class, which is imported automatically by:

```
from scw.core.target import Target
```

Two different types of target can be defined: scalar targets and xy profile targets. See Target on page 129.

Scalar Targets

The most basic targets are scalar targets, which correspond to numeric entities that are extracted variables in Sentaurus Workbench.

An extracted variable in Sentaurus Workbench and its extracted numeric value appear in a simulator log file in the following expression on a new line:

```
DOE: <variable> <value>
```

For such extracted variables present in the parent project, a target is defined by:

```
t1 = Target(name=<s>, value=<f>)
```

where `name` is the name of a Sentaurus Workbench extracted variable of the scalar target, and `value` is the numeric goal value of the scalar target.

For example, for a topography calibration, an extracted variable of a critical dimension (CD) called `BottomZ` appears in the log file of Sentaurus Topography 3D with a value of 3.4567 μm as follows:

```
DOE: BottomZ 3.4567
```

The corresponding target object named `t_BottomZ` for the extracted variable `BottomZ` with a goal value of 4 μm must then be defined by:

```
t_BottomZ = Target(name="BottomZ", value=4.0)
```

XY Profile Targets

The xy profile targets correspond to xy curves or profiles. For such xy profiles present in the parent project, a target is defined by:

```
t2 = Target(name=<s>, value=<s>, simulation_variables=<t>,
            reference_variables=<t>)
```

where `name` is the name of a Sentaurus Workbench extracted variable of the simulated output file of the profile target, and `value` is the data file including the reference target xy profile. The argument `simulation_variables` requires a list of two arguments, which specify the x- and y-variables of the simulation output profile. Similarly, the argument `reference_variables` requires a list of two arguments, which specify the x- and y-variables of the reference profile target.

For example, for a device calibration, the target named `t_IdVg` for an I_d–V_g curve in the file `IdVg.plx` with x-variable `Vg` and y-variable `Id`, which correspond to the Sentaurus Device output file extracted to the variable `IdVgFile` and with x-variable `Gate OuterVoltage` and y-variable `Drain TotalCurrent`, can be defined by using:

```
t_IdVg = Target(name="IdVgFile", value="IdVg.plx",
           reference_variables=['Vg', 'Id'],
           simulation_variables=['Gate OuterVoltage', 'Drain TotalCurrent'])
```

NOTE To create the extracted variable of the simulated output file of a profile target, you should write it explicitly to the log file by using a logging command in the simulator command file.

For Tcl input command files (for Sentaurus Process, Sentaurus Topography 3D, and Sentaurus Device in Tcl mode), you can do this, for example, by specifying:

```
puts "DOE: PltFile n@node@_des.plt"
```

For typical Sentaurus Device command files, you can do this, for example, by specifying:

```
Solve {
   System("echo \"DOE: PltFile n@node@_des.plt\" ")
   ...
}
```

Split Conditions

If you define split conditions, that is, condition parameters that can have different values, then the target definition must be linked to specific conditions.

For both scalar and xy profile targets, you must specify the argument `conditions` as follows:

```
t1 = Target(name=<s>, value=<f>, conditions=<d>)

t2 = Target(name=<s>, value=<s>, conditions=<d>,
            simulation_variables=<t>, reference_variables=<t>)
```

The argument `conditions` specifies a dictionary of condition parameter names and their values.

For example, for a topography calibration with different etching times to be calibrated, the target `t_BottomZ` for a CD with the extracted variable name `BottomZ` has the goal value of 4 μm in the case of an etching time of 30 minutes. This must be specified by:

```
t_BottomZ = Target(name="BottomZ", value=4.0, conditions={"time": "30"})
```

In this case, it is also recommended to specify a unique ID for the target, which is used as the identifier internally and for logging purposes. This is specified by:

```
t_BottomZ = Target(name="BottomZ", id="t30_BottomZ", value=4.0,
                   conditions={"time": "30"})
```

Moreover, for example, for a device calibration with different drain voltages and device types, the target `t_IdVg` for an I_d–V_g curve in the file `IdVg.plx` is for the device type `NMOS` and a drain voltage of 2.0 V. This must be specified by:

```
t_IdVg = Target(name="IdVgFile", value="IdVg.plx",
           conditions={'device': 'NMOS', 'Vd': '2,0'}
           reference_variables=['Vg', 'Id'],
           simulation_variables=['Gate OuterVoltage', 'Drain TotalCurrent'])
```

In these cases, the specified values for the condition parameters are used for the simulations corresponding to these targets. If no conditions are specified in the target definition, then the default values are used for the condition parameters. The default values are taken from the nominal experiment of the parent project. The nominal experiment of the parent project is specified by the `parent_experiment` argument (see Defining the Parent Project and Simulation Tool Flow on page 14).

Excluding and Including Targets in Calibration Tasks

When targets are defined, they should be assigned to a Python variable. To be included in subsequent calibration tasks, the target (object) variable should be included in a target (object) list:

```
targets = [t1]
```

Multiple targets can be included in a target list:

```
targets = [t1, t2, t3]
```

If a target should *not* be included in a task, then you can exclude it from the target list of the subsequent calibration tasks.

Setting a Weight for Targets

You can set several control options for a target when it is defined. For example, each calibration target can have a weight that is applied when aggregating different targets. The weight is specified as follows:

```
t1 = Target(name=<s>, value=<f>, weight=2.0)
```

In this case, the error of the given target is twice as important as the error of other targets without any weight specification, since the default weight is 1.0.

Setting an Error Method for Targets

Each target has an individual method to calculate the pointwise error between the simulation response and the reference value. The following command sets the error method:

```
t1 = Target(name=<s>, value=<f>, error_method=<s>)
```

For details about the available error methods and other arguments, see Target on page 129.

For example, for a device calibration with a target for an I_d–V_g curve, it can be useful to calibrate against the xy profile with a wide range of magnitude in logarithmic scale. For example:

```
t_IdVg = Target(name="IdVgFile", value="IdVg.plx",
           reference_variables=['Vg', 'Id'],
           simulation_variables=['Gate OuterVoltage', 'Drain TotalCurrent'],
           error_method="log_abs")
```

To restrict the window of a target profile for comparison, use the following command to define the lower and upper bounds of the x– and y–data range:

```
t2 = Target(name=<s>, value=<s>, conditions=<d>,
            simulation_variables=<t>, reference_variables=<t>,
            x_range=<t>, y_range=<t>)
```

For example, for a topography calibration with a target for an etch contour, where a relative error is used, it is necessary to exclude zero values of the reference target xy profile. Moreover, the focus might be only on a part of the etch contour, for example, the negative values of the reference target xy profile. In this case, the ranges are specified in the following way:

```
XYBowingCD = Target(name='XYBowingCD', value='/reference.csv',
                    simulation_variables=['Z', 'Radius_Avg'],
                    reference_variables=[' Z', ' Radius_Avg'],
                    error_method="rel",
                    x_range=[1e30, 0], y_range=[1e-10, 1e30])
```

Attributes for Targets

You can retrieve the following attributes of a `Target` object:

- `name`: Name of the Sentaurus Workbench parameter or variable in which the simulation response is available
- `value`: Value to be matched by the simulation response

- `id`: Unique name or ID of the target
- `type`: Type for target classification
- `dtype`: Datatype of the target
- `conditions`: Condition or split parameters that must be set to calculate the calibration targets
- `simulation_variables`: Specifies a tuple with the names of the independent (x) and dependent (y) variables (in that order) in the simulation file
- `reference_variables`: Specifies a tuple with the names of the independent (x) and dependent (y) variables (in that order) in the reference file

You can retrieve and set the following attributes of a `Target` object:

- `weight`: Weight to be applied to the calibration target
- `scale`: Scaling factor used to normalize the target value
- `transform`: Optional transform function to be applied to the target
- `penalty`: Penalty to be applied to the target if the constraints are violated
- `x_range`: Range of interest for the independent (x) variable
- `y_range`: Range of interest for the dependent (y) variable
- `x_scale`: Scaling factor for the independent (x) variable
- `y_scale`: Scaling factor for the dependent (y) variable
- `error_method`: Method used to calculate the pointwise error between the simulation response and the reference value

For example, to retrieve the current value of the target `tar`, use the variable:

```
tar.value
```

To update the current weight, use the assignment:

```
tar.weight = <f>
```

Setting Options for Calibrations

You can define several optional calibration controls by using the `set_option()` command. Some controls are explained here. For details about this command and its arguments, see set_option on page 127.

Defining the Parent Project and Simulation Tool Flow

The default parent project, which includes the simulation flow, is defined by:

```
cal.set_option(parent="/path/of/parent/project")
```

If no parent project is specified for a particular calibration task, then the default parent project is used.

If the parent project consists of more than one experiment, then the nominal experiment must be specified by:

```
cal.set_option(parent_experiment=<i>)
```

The nominal experiment defines all the condition values unless they are specified explicitly in the target definition (see Split Conditions on page 10).

The simulation tool flow is specified as a list of tool labels. For example, for a flow with three tool labels, `sprocess`, `sdevice`, and `svisual`, it is defined by:

```
cal.set_option(tools=["sprocess", "sdevice", "svisual"])
```

Only the relevant tool steps need to be included; all others will be *bridged* to include their parameters, but they are not executed. However, the Bridge tool cannot be placed in between tools.

After creation of the child project, the command and preference files of the specified tool labels are copied from the parent project.

A list of additional files and folders of the parent project to be copied to the child project (such as sourced files with target data or simulation parameters) is defined, for example, for the file `file.txt` and the folder `folder`, by:

```
cal.set_option(files_to_copy=["file.txt", "folder"])
```

The file and folder names can include wildcards, for example, if all files with a specific file extension must be copied:

```
cal.set_option(files_to_copy=["*.fps"])
```

The list can also include files and folders outside of the parent project, for which the path should be specified:

```
cal.set_option(files_to_copy=["/path/of/file.txt", "/path/of/folder"])
```

Defining Extra Tool Parameters

In the case of calibration parameters that are not included in the parent project, they can be added to the child project by specifying:

```
cal.set_option(extra_tool_parameters=<s|d>)
```

The `extra_tool_parameters` argument specifies the tool labels to which missing Sentaurus Workbench parameters are added.

If it specifies a single string, then all parameters are added to this tool label. If it specifies a dictionary, then the parameters are added to individual tool labels.

This feature is usually used in combination with copying a calibration or parameter file, which is parameterized with the Sentaurus Workbench parameters to be added, for example, a Sentaurus Device parameter file `sdevice.par`, which includes model parameterization. This file must be copied by using the argument `files_to_copy` (see Defining the Parent Project and Simulation Tool Flow on page 14).

Defining the Child Project Directory

The default child project directory in which Sentaurus Calibration Workbench generates all child projects is defined by:

```
cal.set_option(child_dir="/path/to/child/project/directory")
```

If no child project directory is specified for a particular calibration task, then the default child directory is used.

Child projects are named after parent projects and are placed into child subdirectories following the name of the Sentaurus Calibration Workbench project. The final subdirectory will be named `<child_dir>/<scw_project>/<parent_name>`.

Setting the Simulation Job Queue

The job queue to which the simulation should be submitted is defined by:

```
cal.set_option(queue=<s>)
```

Calculating the Total Weighted Root Mean Square Error

The way of calculating the total weighted RMS error is defined by:

```
cal.set_option(use_scalib_weights=<b>)
```

The total weighted RMS error for N individual target errors e_i and weights w_i is calculated as follows:

- If `use_scalib_weights=True`: $\text{RMS} = \sqrt{\frac{\sum (w_i \times e_i)^2}{\sum (w_i)^2}}$
- If `use_scalib_weights=False`: $\text{RMS} = \sqrt{\frac{\sum w_i \times e_i^2}{N}}$

Setting the Maximum Number of Iterations

For a gradient-based calibration algorithm, the maximum number of iterations per calibration task can be specified by:

```
cal.set_option(max_iteration=50)
```

Performing Calibrations

To perform a calibration, you execute one or more calibration tasks, including:

- Simple Evaluation
- Gradient-Based Local Calibration
- Design-of-Experiments Search
- Sensitivity Analysis
- Calibration Based on Machine-Learning Models
- Calibration by Exploring Machine-Learning Models
- Calibration by Refining Machine-Learning Models
- Validating Parameter Ranges

In the case of a sequence of calibration tasks, each task is assigned a step signature. To retrieve the current calibration step or task signature, use the command:

```
cal.get_step_signature()
```

Simple Evaluation

To run a simple evaluation task for all included targets with the current parameter values without any calibration, use the command:

```
cal.evaluate(parameters=<t>, targets=<t>)
```

This command is required for an initial evaluation needed to register all nominal or initial parameter values and to validate the simulation setup as well as the initial accuracy. Moreover, the command is useful for an intermediate evaluation or a final evaluation in the calibration flow.

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.evaluate(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see evaluate on page 103 and set_option on page 127.

Gradient-Based Local Calibration

In the simplest situation, you have a set of parameters that you know is a good initial guess for the calibration process. You can then use a gradient-based local calibration algorithm (also known as *optimization*) to find the best set of parameters by calling:

```
cal.optimize(parameters=<t>, targets=<t>)
```

This command runs a calibration of all included parameters against all included targets. For details about how to determine the termination of calibration loops, see Terminating Calibration Loops on page 18.

When it is not possible to obtain a good calibration result for a single calibration task, multiple calls of `cal.optimize()` are possible as a sequence of calibration tasks. The following calibration task starts with the calibrated parameter values from the previous calibration task. You can change the included parameters and targets in between the calibration tasks.

To set the maximum number of iterations per calibration task, either use the optional argument `max_iteration`, for example:

```
cal.optimize(parameters=<t>, targets=<t>, max_iteration=50)
```

or use the command:

```
cal.set_option(max_iteration=50)
```

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.optimize(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see optimize on page 112 and set_option on page 127.

Setting Controls for the Backend

The default gradient-based local calibration algorithm is called `scalib`, which is a backend specific to Sentaurus Calibration Workbench and is based on the Levenberg–Marquardt algorithm.

Optionally, you can set controls for calibration parameters as follows:

```
cal.set_backend_control([initdx=<f>,] [mindx=<f>,] [maxdx=<f>])
```

These controls include `initdx` for the initial normalized parameter update for calibration, `mindx` for the minimum of the normalized parameter update, and `maxdx` for the maximum of the normalized parameter update.

In addition, you can optionally set calibration convergence criteria for the `scalib` backend as follows:

```
cal.set_backend_control([ssq_error_tolerance=<f>,]
                        [parameter_tolerance=<f>,]
                        [target_tolerance=<f>])
```

These controls for convergence criteria include `ssq_error_tolerance` for the relative change in the sum of squares (ssq), `parameter_tolerance` for the change in the norm of the parameter vector, and `target_tolerance` for the tolerance of the total target error (see Terminating Calibration Loops).

Terminating Calibration Loops

The termination of calibration loops for the `scalib` backend is determined, in order, by the following:

1. When the total weighted RMS error is less than `target_tolerance` (default: 0.0). See Setting Controls for the Backend on page 18.

2. When the ssq value of the target errors increases or the decremental ratio of the ssq value for targets is less than `ssq_error_tolerance` (default: 1e-5), and the ssq value of parameter variations is less than `parameter_tolerance` (default: 2e-3). See Setting Controls for the Backend on page 18.
3. When the looping number exceeds the maximum number (`max_iteration`) specified in the `optimize()` command or, if not present, in the `set_option()` command. The default is 50. However, for more challenging calibration problems, a larger `max_iteration` is recommended.

Design-of-Experiments Search

To explore the parameter space, running and evaluating a design-of-experiments (DoE) can be helpful. You can use the following command to explore the input space of parameters and to have a better idea of how the target responses change with the value of the parameters:

```
cal.search(parameters=<t>, targets=<t>, method=<s>, options={"Ns": <i>})
```

Here, `method` defines the DoE method, and the option `Ns` defines the number of experiments in the DoE in addition to the current nominal parameter set.

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.search(parameters=<t>, targets=<t>, method=<s>, options={"Ns": <i>},
           parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see search on page 124 and set_option on page 127.

Sensitivity Analysis

Sensitivity analysis is used to understand how the variability of parameters affects simulation results and accuracy. In particular, one-parameter-at-a-time sensitivity analysis aims at understanding the response of a model as a function of small changes to one parameter, with all other parameters fixed. Therefore, it reveals only the local gradient of the response surface of the model with respect to a given parameter.

To compute the sensitivity for a list of parameters and targets, specify:

```
cal.run_sensitivity_analysis(parameters=<t>, targets=<t>)
```

The increment (positive ∂x) or decrement (negative ∂x) of the normalized parameter value to calculate the sensitivity is defined by `sensdx` in the `set_backend_control()` command by default (see Setting Controls for Parameters on page 21). The normalized parameter update can also be set directly by:

```
cal.run_sensitivity_analysis(parameters=<t>, targets=<t>,
                             percentage_range=<f>)
```

By default, the sensitivity is computed bidirectionally for positive and negative ∂x, which corresponds to 3 points on the parameter axis. To compute the sensitivity monodirectionally for positive ∂x only, specify the number of points explicitly:

```
cal.run_sensitivity_analysis(parameters=<t>, targets=<t>, num_points=2)
```

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.run_sensitivity_analysis(parameters=<t>, targets=<t>, parent=<s>,
                             child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see run_sensitivity_analysis on page 121 and set_option on page 127.

Based on a computed sensitivity, you can determine the primary parameters by using:

```
primary_parameters = cal.determine_primary_parameters(parameters=<t>,
                                                      num_parameters=<i>,
                                                      min_sensitivity=<f>)
```

This command returns a list of only primary parameters for a subsequent calibration task, excluding secondary parameters.

For details about this command and its arguments, see determine_primary_parameters on page 101.

Setting Controls for Parameters

Optionally, you can set controls for the normalized parameter update for sensitivity analysis as follows:

```
cal.set_backend_control(sensdx=<f>)
```

Calibration Based on Machine-Learning Models

To perform a calibration strategy, based on machine learning, on a given set of parameters for a given set of targets, use the command:

```
cal.calibrate_ml(parameters=<t>, targets=<t>)
```

The machine learning–based calibration strategy consists of a sequence of machine-learning model exploration and refinement subtasks, and is designed for general calibration problems.

To set the parent project and the child project directory of the calibration task, you can use the optional arguments `parent` and `child_dir`, for example:

```
cal.calibrate_ml(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see calibrate_ml on page 99 and set_option on page 127.

For more information about low-level functions for machine learning, see Chapter 3, Sentaurus Calibration Workbench Machine-Learning Module.

Calibration by Exploring Machine-Learning Models

To execute an exploration of a machine-learning model on a given set of parameters for a given set of targets, use the command:

```
cal.explore_ml(parameters=<t>, targets=<t>, num_iterations=<i>)
```

The calibration parameters are calibrated based on the exploration algorithm, which works as follows:

1. A child project for machine-learning training is created and run.
2. Machine-learning models are created for each target and split condition. A machine-learning model database is populated.
3. The machine-learning database is used to predict calibration candidates. These candidates are simulated, and the best candidate is selected.
4. The exploration is repeated until the maximum number of iterations (`num_iterations`) is reached.

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.explore_ml(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see explore_ml on page 108 and set_option on page 127.

For more information about low-level functions for machine learning, see Chapter 3, Sentaurus Calibration Workbench Machine-Learning Module.

Calibration by Refining Machine-Learning Models

To execute a refinement of a machine-learning model on a given set of parameters for a given set of targets, use the command:

```
cal.refine_ml(parameters=<t>, targets=<t>, num_iterations=<i>)
```

The calibration parameters are calibrated based on the refinement algorithm, which works as follows:

1. A child project for machine-learning training is created and run.
2. A machine-learning model is trained on data defined around the current nominal value and is used to predict new candidates. These candidates are simulated, and the best candidate (experiment) is selected.
3. Several iterations are performed until the maximum number of iterations (`num_iterations`) is reached or the convergence criteria are met.

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.refine_ml(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see refine_ml on page 119 and set_option on page 127.

For more information about low-level functions for machine learning, see Chapter 3, Sentaurus Calibration Workbench Machine-Learning Module.

Validating Parameter Ranges

Parameter ranges are critical for the creation of a machine-learning model. If the parameter exploration space is too large, then the combination of parameters could lead to failures in simulations and, subsequently, to insufficient data for model training.

The following command executes the validation of the machine learning–based parameter range:

```
cal.validate_range_ml(parameters=<t>, targets=<t>, constraint=<s>)
```

The validation algorithm works as follows:

1. A sensitivity analysis is executed with the full parameter range. Sensitivity includes upper and lower parameter values.
2. The parameter range is decreased or increased if there is a failure in the simulations. If the failure occurs for the lower value, then the parameter lower value is increased. If the failure occurs for the upper value, then the parameter upper value is reduced.
3. Failed experiments include those that match the constraint condition specified by `constraint` or if the RMS error is above a certain threshold.
4. If the parameters have reached a range that is too close to the nominal or if the number of internal iterations is reached, then the algorithm stops.

The log file shows the resulting optimum, lower, and upper values for each parameter. If the value was changed by the validation range task, then the corresponding *previous* value will differ.

To set the parent project and the child project directory of the calibration task, either use the optional arguments `parent` and `child_dir`, for example:

```
cal.validate_range_ml(parameters=<t>, targets=<t>, parent=<s>, child_dir=<s>)
```

or use the command:

```
cal.set_option(parent=<s>, child_dir=<s>)
```

For details about these commands and their arguments, see set_option on page 127 and validate_range_ml on page 133.

Calibration Output

This section discusses the calibration output.

Sentaurus Workbench Child Projects

Upon execution, Sentaurus Calibration Workbench creates one or more separate Sentaurus Workbench projects that include only the tool instances that are necessary for the calibration, adding all other tool instances to a Bridge tool instance. This automatically created child project includes the experiments for each evaluation or iteration, and it is run in the background.

NOTE By default, child projects are not deleted when cleaning up the Sentaurus Calibration Workbench node or output project in Sentaurus Workbench. If the child project is already present and consistent with the current run, then it will be reused including the simulation results if present.

Calibrated Parameters

The resulting calibration parameter values can be found in the log file of Sentaurus Calibration Workbench. In addition, you can export them to the Sentaurus Workbench variables table or to parameter files.

The following command writes the calibrated parameters to the log file in the syntax for Sentaurus Workbench variable extraction:

```
cal.print_parameters(parameters=<t>, [suffix=<s>])
```

The optional argument `suffix`, which is `_cal` by default, specifies the suffix added to parameter names for Sentaurus Workbench variable extraction, in order to differentiate them from Sentaurus Workbench parameter names. See print_parameters on page 115.

Calibrated Targets

The log file of Sentaurus Calibration Workbench contains the resulting target evaluation values and RMS errors. In addition, you can export them to the Sentaurus Workbench variables table or to parameter files.

The following command writes the calibrated target values to the log file in the syntax for Sentaurus Workbench variable extraction:

```
cal.print_targets(targets=<t>, [suffix=<s>,] [include_path=<b>])
```

The optional argument `suffix`, which is `_cal` by default, specifies the suffix added to parameter names for Sentaurus Workbench variable extraction, in order to differentiate them from preexisting Sentaurus Workbench variable names. Finally, in the case of files as target values, the full path of that file can be included by setting the `include_path` argument to `True`. See print_targets on page 116.

Calibration History

To create a history plot named `error.png`, which includes the error history of a calibration, use the following command *after* the calibration tasks:

```
cal.plot_error_history(file="error.png")
```

If you also want to include the individual target errors in the history plot, then include the target list in the command:

```
cal.plot_error_history(file="error.png", targets=<t>)
```

See plot_error_history on page 114.

Calculating the Root Mean Square Error

Sentaurus Calibration Workbench calculates the RMS error between each evaluated value and target value. For details about the calculation of the target RMS error, see Targets on page 8.

In addition, the total weighted RMS error is calculated, taking into account the weights of each target.

RMS errors are printed to the log file for each evaluation. In addition, RMS errors can be accessed for each target by specifying:

```
cal.get_rms_error(target=<s>)
```

To retrieve the current total weighted RMS error of all targets, use the command without any arguments:

```
cal.get_rms_error()
```

See get_rms_error on page 110.

CHAPTER 3 Sentaurus Calibration Workbench Machine-Learning Module

This chapter describes how to use the Sentaurus Calibration Workbench machine-learning module to create and train machine-learning models.

Introduction to Machine Learning

The Sentaurus Calibration Workbench machine-learning (SCWML) module provides methods to collect and process TCAD data and generate machine learning–based models with prediction capabilities. The SCWML module is used in Sentaurus Calibration Workbench for machine-learning applications.

The SCWML module allows you to define TCAD experiments and parameters, the design-of-experiments (DoE) method, the machine-learning input (feature) preprocessing method, and the machine-learning model. Most of the hyperparameters (settings and options in machine-learning models) are open for you to define.

The SCWML module also provides reasonable default values for the hyperparameters. The prediction capabilities of the machine-learning model allow you to apply the model to different purposes such as gradient-based local calibration, screening, and subsequent TCAD simulations.

Basic Machine-Learning Procedure

The basic machine-learning procedure includes the following steps:

1. Data generation: prepare the training set, with feature-output data points (alternatively, the dataset can be loaded from an external file)
2. Training: fit a model based on the training set
3. Inference (prediction): the model predicts new outputs from new features

Several popular add-ons and variations have been derived from this basic procedure. You can add a validation step to the procedure, which uses the original TCAD tools to obtain the true output values and to compare them with the machine-learning inference.

Often, you want to go through more than one training–validation–evaluation iteration for fine-tuning hyperparameters. Another variation is *adaptive data generation*, in which, in contrast to a single data generation–training–inference flow, data generation and training are repeated iteratively, to generate data targeting certain output criteria.

With a trained model, you can also perform a reverse search, which finds the combination of parameters that yields targeted output values.

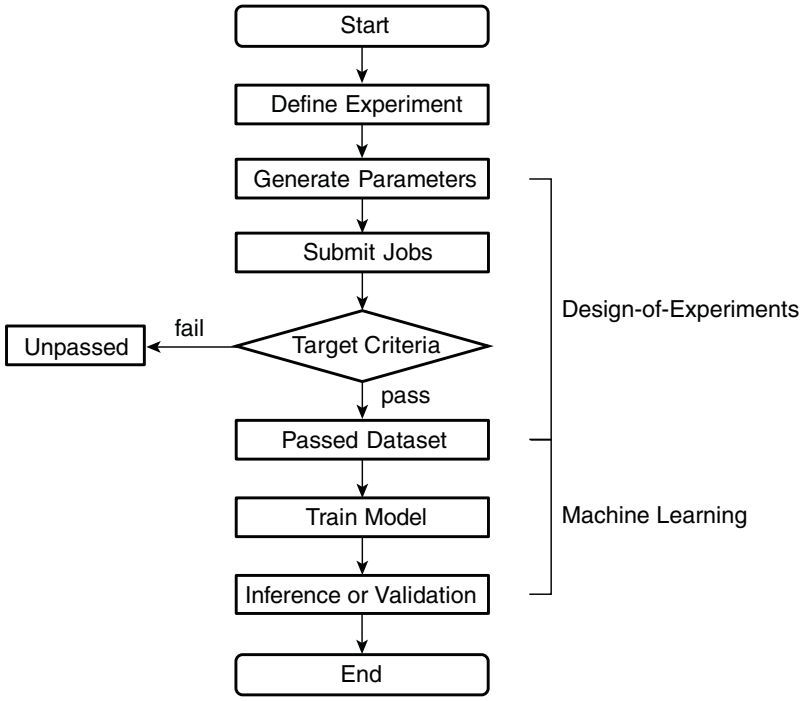

Figure 1 Flowchart of a simple machine-learning process

The SCWML module includes the basic machine-learning procedure with validation and adaptive data generation. It also supports loading data from an external source. In the remainder of this chapter, you go through the following steps:

- Setting Up the Machine-Learning Object
- Data Generation With Design-of-Experiments Methods
- Setting Up Child Project Options
- Data Generation With Adaptive Data Generation Method
- Setting Up Machine-Learning Model and Training
- Inference
- Model Persistence
- Reverse Search

You will work with a machine-learning object, which stores the relevant information to reproduce the full TCAD and machine-learning procedure.

Prerequisites

The following instructions assume that the correct libraries have been imported into the environment. Specifically:

```
from scw.ml import SCWML
from scw.core.parameter import Parameter
from scw.core.target import Target
from scw.core.scwglobal import *
```

You should be familiar with basic Python syntax and Sentaurus Workbench value substitution and extraction mechanisms:

- In the template script, @-enclosed parameters (`@par@`) are substituted by the value of `par`.
- During simulation execution, standard output in the format `DOE: <variable> <value>` allows Sentaurus Workbench to extract the value of the variable `<variable>` as `<value>`.

Using Different Tool Versions

The SCWML module manages the child project through Sentaurus Workbench, whose version can differ from the version of Sentaurus Calibration Workbench.

To use a different version of Sentaurus Workbench:

1. In the shell to run Sentaurus Calibration Workbench, set the environment variable `STRELEASE_SWBPY` to the Sentaurus Workbench version:

   ```
   setenv STRELEASE_SWBPY <swb_version>
   ```

   For example:

   ```
   setenv STRELEASE_SWBPY Q-2019.12
   ```

2. In the script, add the setting:

   ```
   obj.set_project(..., gsub_command="gsub -rel <swb_version>",
                   spp_command="spp -rel <swb_version>")
   ```

3. To specify the Sentaurus Calibration Workbench version, run the following command in the shell:

   ```
   scw -rel <scw_version>
   ```

Alternatively, to specify the Sentaurus Calibration Workbench version:

a) In Sentaurus Workbench, select Sentaurus Calibration Workbench in the tool row.

b) Choose **Tool > Properties** or double-click the tool icon.

The Tool Properties dialog box for Sentaurus Calibration Workbench opens.

c) Click the **Tool Properties** tab.

d) Enter `-rel <scw_version>` in the **Command Line** field.

e) Click **OK** or **Apply**.

Setting Up the Machine-Learning Object

The SCWML object manages the feature-output relation of parameters and variables by using the `Parameter` and `Target` objects. Before their initialization, parameters and targets must be defined.

The parameters to be included as machine-learning model inputs are defined by:

```
<parameter_variable> = Parameter(name=<s>, value=<s>|<i>|<f>,
                                 [minimum=<i>|<f>, maximum=<i>|<f>,]
                                 [transform=<o>|None,] [choices=<t>,]
                                 [sort=<b>,] [dtype=<k>,] [force_pass=<b>])
```

where:

- `parameter_variable` is a Python variable name for the parameter object.
- `name` is the name of the parameter, which together with the variable name is used to identify the parameter.
- `value` is the initial value or the default of the parameter.
- `minimum` and `maximum` are the lower and upper bounds of the valid values for a continuum parameter.
- `transform` is the numeric transformation of the variable values before passing to the machine-learning model. If set to `None` (default), then no transformation is performed.
- `choices` is a list of valid values for a discrete parameter.
- `sort` specifies whether to sort the discrete values before passing them to the machine-learning model.
- `dtype` sets the Python datatype of the parameter. Valid options include `float` (default), `bool`, `int`, or `str`.
- `force_pass` specifies whether, during data sampling, instead of using the DoE method, the experiments are forced to pass every value given by `choices` for this parameter. `force_pass` parameters are sometimes referred to as *conditions*.

The `choices` values for a parameter might be duplicate, when values of other `force_pass` parameters differ. For example:

Consider a project with `temperature` and `time` as `force_pass` parameters (conditions), and three condition splits of `temperature` and `time` must be considered:

```
time=10, temperature=300
time=10, temperature=400
time=30, temperature=300
```

The condition split of `time=30, temperature=400` is not available.

The condition splits are registered by:

```
p_time = Parameter(name="time", choices=[10, 10, 30], force_pass=True)

p_temp = Parameter(name="temperature", choices=[300, 400, 300],
                   force_pass=True)
```

The variables `p_time` and `p_temp` are passed to the main machine-learning object during its initialization. At data generation, the non-`force_pass` parameters are sampled using a DoE method (Box–Behnken or random sampling, for example), while each of the generated combinations of parameter values takes all three `time–temperature` condition splits together as simulation inputs.

Targets, the output of the simulation, can be scalar values or a set of values representing curves. Targets are defined by:

```
<target_variable> = Target(name=<s>,
                           range=<t>, abs_value_range=<t>,
                           values=<t>, dimension=<i>, dtype=<k>)

<target_variable> = Target(name=<s>,
                           simulation_variables=<t>,
                           x_range=<t>, y_range=<t>, dtype=<k>)
```

where:

- `name` is the name of an extracted variable created in the Sentaurus Workbench project. If the target is scalar, then the extracted value is the actual value of the target. If the target is a curve, then the extracted value is the path and name to the file containing the curve values.
- The passing scalar target criteria are optional:
  - `range=[<min>, <max>]` sets the continuous range of acceptable values for the target. To set only the minimum (or maximum) value, `<max>` (or `<min>`) can be set to `None`.
  - `abs_value_range=[<min>, <max>]` sets the continuous interval of acceptable values for the absolute value of the target. Similarly, one of the bounds can be set to `None`.
  - `values` specifies a discrete list of acceptable values for the target.

If any target criteria are defined, then they are used for the following purposes:

- During machine-learning training, you can choose to train the model based on the full dataset or a subset with outputs (targets) satisfying these criteria.
- If you choose the adaptive data generation method, then a parameter filter is trained to gradually generate more data satisfying the criteria.

- `dimension` must be set to `2` for curve targets.
- `simulation_variables` sets the names of the x-axis and y-axis to extract for the curve file.
- `x_range=[<min>, <max>]` and `y_range=[<min>, <max>]` set the limits of the extracted curve.
- `dtype` specifies the parameter datatype. Options are `float`, `int`, and `str`. If it is not provided, then it is a Python `float` by default.

The simulation flow from the parameters to the targets are defined by a parent project. The machine-learning object is initialized from the parent project by:

```
obj = SCWML([parent=<s>,] [db_name=<s>|None,]
            [parameters=<t>|None,] [targets=<t>|None])
```

where:

- `parent` defines the root path of the parent project (default: `"."`).
- `db_name` defines the name of the database. If it is not provided, then a generic name is used.
- `parameters` is a list of parameter variables to be included in the machine-learning model.
- `targets` is a list of target variables to be included in the machine-learning model.

You exclude some parameters from the machine-learning model by specifying:

```
obj.exclude_parameters(<s>, [<s>, ...])
```

where parameter names to exclude are the arguments of the command. By excluding some setting-type parameters, the machine-learning model can focus on more relevant input features.

NOTE The `exclude_parameters()` command can be called before and after the data generation step. If it is called *before* data generation, then the excluded parameter and condition values are not recorded in the database and cannot be recovered.

In some applications, a single simulation can produce a series of input–output data points, for example, a series of etching profiles at different time steps. To split a single simulation into a set of data points, the series of parameter names and target names must be merged into one. Name-merging is performed by:

```
obj.merge_feature_target_names(table=<dict>)
```

Here, `table` is a dictionary with the name-merging rules.

For example, consider a simulation with three time steps `time_1`, `time_2`, `time_3` and the respective profiles `profile_1`, `profile_2`, `profile_3`. To merge the time steps into a parameter `time` and the profiles into a parameter `profile`, the names must be defined first:

```
t1 = Parameter("time_1", ...)
t2 = Parameter("time_2", ...)
t3 = Parameter("time_3", ...)
t = Parameter("time", ...)

p1 = Target("profile_1", ...)
p2 = Target("profile_2", ...)
p3 = Target("profile_3", ...)
p = Target("profile", ...)
```

The SCWML object is initialized with all the defined names:

```
obj = SCWML(parent, parameters=[t1, t2, t3, t], targets=[p1, p2, p3, p], ...)
```

Then, the name-merging rules can be specified by:

```
obj.merge_feature_target_names(table={"time":["time_1", "time_2", "time_3"],
                           "profile":["profile_1", "profile_2", "profile_3"]})
```

NOTE Name-merging occurs at the moment before the machine-learning model training. During data generation, options and controls are applied to the premerged names. After training, for prediction and reverse search, merged names are used.

Data Generation With Design-of-Experiments Methods

Data generation is available after finalizing the experiment. The supported design-of-experiments (DoE) data generation methods are:

- Full-factorial
- Box–Behnken
- Latin hypercube
- Sobol sampling
- Random uniform
- Adaptive data generation

Figure 2 shows the commands used to set up and generate the dataset (commands in the dashed box are optional and relate to the adaptive data generation method).

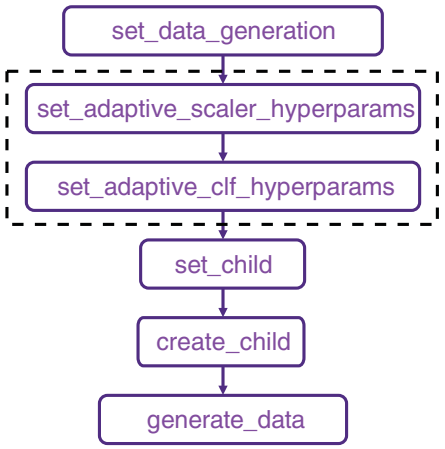

Figure 2 Commands used to set up and generate the dataset; commands in dashed box are specific to the adaptive data generation method

The settings for data generation are controlled by:

```
obj.set_data_generation(doe=<s>, num_samples=<i>, random_seed=<i>)
```

The `doe` argument defines the data generation method. Other arguments depend on the chosen method:

- The full-factorial method (`doe="full_factorial"`) and the Box–Behnken method (`doe="box_behnken"`) do not require any additional arguments.
- The Latin hypercube method (`doe="latin_hypercube"`), the Sobol sampling method (`doe="sobol"`), and the random uniform method (`doe="random"`) require the size of the dataset (`num_samples`) and a random seed (`random_seed`).

  NOTE Every parameter combination must pass through all condition splits. Therefore, the real number of samples will be `num_samples` times the condition split multiple.

- For details about the adaptive data generation method (`doe="adaptive"`), see Data Generation With Adaptive Data Generation Method on page 35.

Setting Up Child Project Options

The SCWML module manages the experiment jobs through a child project. The Sentaurus Workbench child project is set up by:

```
obj.set_child(child=<s>, queue=<s>, tools=<t>,
              [files_to_copy=<t>,] [gsub_command=<s>,] [autosave_file=<s>])
```

where:

- `child` is the path to a child project (default: `"../internal"`).
- `queue` sets the queue to which the child project submits jobs (default: `"local:default"`).
- `tools` lists the labels of the steps from the parent project to be added to the child project.
- `files_to_copy` lists the supporting files to copy to the child project.
- `gsub_command` specifies the command to submit the child project (default: `"gsub"`).
- If `autosave_file` is set, for each batch of the simulation job submission, an autosave `.pkl` file is generated in the path in which Sentaurus Calibration Workbench is called (default: `None`).

Child Project Generation and Submission

After the child project and data generation options are set, the child project and the initial experiments can be created by:

```
obj.create_child(delete_if_exist=<b>)
```

The argument `delete_if_exist` specifies whether to delete the path if it already exists (default: `False`).

The generated project is submitted to be run and to generate a dataset by specifying:

```
obj.generate_data()
```

After each experiment run, the SCWML module uploads the status, parameter, and variable values to the database.

Data Generation With Adaptive Data Generation Method

You can add an adaptive parameter filter to the data generation process. The filter is essentially a machine-learning classifier, which estimates whether a parameter combination will give target values that meet your criteria.

The adaptive data generation method breaks the target dataset size (`num_samples`) into batches of smaller sizes (`batch_size`). After each batch result is returned, the SCWML module updates the filter based on the full passed or failed cases stored, and submits only the experiment candidates that pass the filter to the next batch.

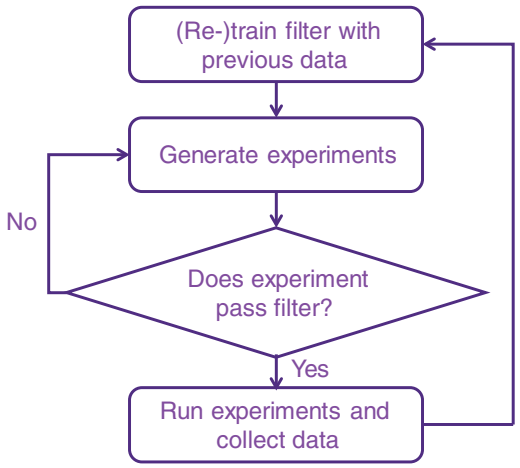

Figure 3 Simplified flow of adaptive data generation method

To use the adaptive data generation method, set `doe="adaptive"` in the following command:

```
obj.set_data_generation(doe="adaptive",
                        batch_size=<i>, pass_rate=<f>,
                        threshold_num_pass=<i>,
                        initial_sampling_doe=<k>,
                        initial_doe_batches=<i>,
                        num_samples=<i>, random_seed=<i>,
                        transformer=<k>, classifier=<k>)
```

where:

- `batch_size` sets the number of data points generated before each update on the parameter filter (default: 1).
- `pass_rate` defines the required passing rate of the filter (default: 1.0).
- `threshold_num_pass` is the threshold number of passing cases to invoke the parameter filter, before which the filter is all-passing. This argument ensures that the dataset contains enough passing data points before a meaningful training (default: 1).
- You can choose a different DoE method (`"box_behnken"`, `"full_factorial"`, `"latin_hypercube"`, or `"sobol"`) in `initial_sampling_doe`, before switching to random uniform sampling with an adaptive parameter filter (default: `"random"`, meaning random uniform sampling throughout data generation). The number of initial sampling batches is specified by `initial_doe_batches` (default: -1). If `initial_doe_batches` is negative, then the method specified by `initial_sampling_doe` is used throughout data generation.

- Unlike other DoE methods, `num_samples` in the adaptive data generation method represents the number of data points collected, *not* submitted. Depending on the parameter range definition, the total number of simulation runs might be larger than `num_samples`.
- `random_seed` sets the random seed to be used to generate experiments (default: 1).
- The SCWML module uses the scikit-learn package [1] for the `transformer` (usually a scaler) and the `classifier`. Table 1 and Table 2 list the supported transformers and classifiers.

Table 1 Supported transformers

Keyword	Transformer
`Identity`	Identity transformer (do nothing)
`Minmax`	Min-max scaler
`Standard`	Mean-standard scaler

Table 2 Supported classifiers

Keyword	Classifier
`AdaBoost`	AdaBoost
`DecisionTree`	Decision tree
`GPC`	Gaussian process
`KNNC`	Nearest neighbors
`MLPC`	Neural network

Each transformer or classifier has different hyperparameters. The hyperparameters and their default values follow the rules of the respective scikit-learn methods. The hyperparameters can be passed by the following commands.

For transformers:

```
obj.set_adaptive_scaler_hyperparams(
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...
)
```

For classifiers:

```
obj.set_adaptive_clf_hyperparams(
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...
)
```

For example, you can select the neural network classifier (`classifier=MLPC`), which has hyperparameters such as `hidden_layer_sizes` and `activation`, each with default values [2].

You can change the values of some hyperparameters by specifying:

```
obj.set_adaptive_clf_hyperparams(hidden_layer_sizes=(32, 16, 4),
                                 activation="tanh",
                                 learning_rate="adaptive")
```

After setting the filter, you use the following commands to generate and run the experiments:

```
obj.create_child(delete_if_exist=<b>)
```

If `delete_if_exist` is set to `True`, then the child project is guaranteed to be created from scratch.

```
obj.generate_data()
```

Two criteria are used to check whether an experiment passes:

- The experiment finishes without error.
- The output falls within the given range (if defined).

Setting Up Machine-Learning Model and Training

The SCWML module supports the scikit-learn package [1]. The machine-learning model includes transform operations for parameters and variables, and an estimator model with regression or classification functions. Table 1 on page 37 lists the supported transformers.

Table 3 Supported regressors

Keyword	Regressor
`GPR`	Gaussian process regressor
`Linear`	Linear regressor
`MLPR`	Neural network regressor
`Quadratic`	Quadratic regressor

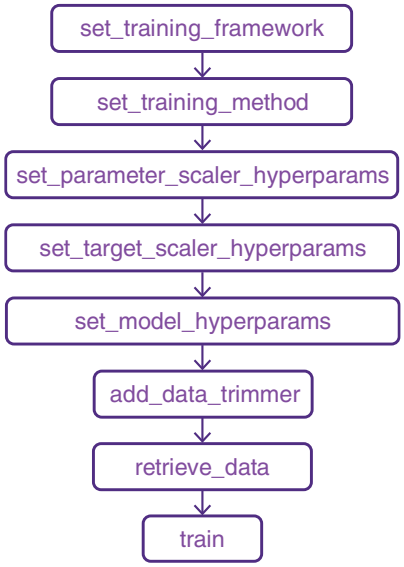

Figure 4 Commands used in the training process

In general, TCAD machine-learning problems are many-to-many problems, which can be described by either one many-to-many model or a collection of many-to-one models (see Figure 5).

NOTE The SCWML module allows both approaches for regression problems, but only many-to-one models for classification problems.

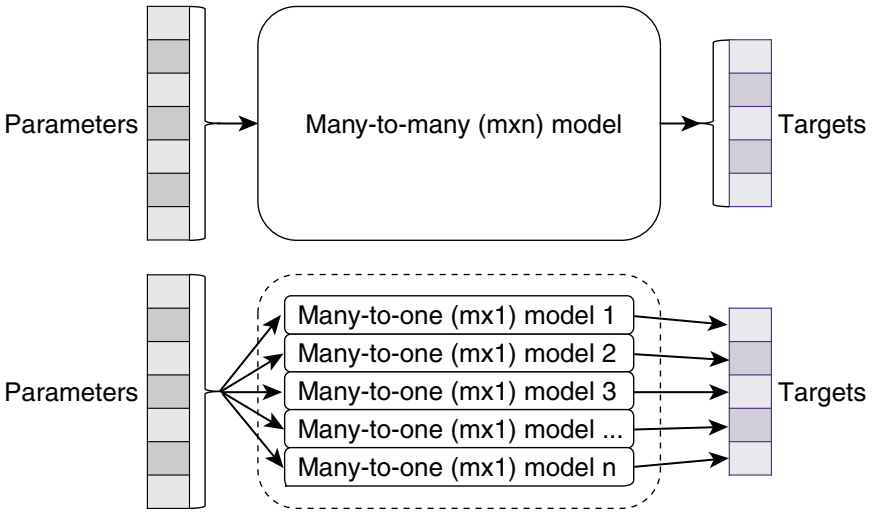

Figure 5 (*Top*) Many-to-many model and (*bottom*) many-to-one models

Many-to-Many Framework

The many-to-many framework is activated by:

```
obj.set_training_framework(framework=MxN, retrain=<b>)
```

NOTE Every command introduced in this section has the argument `retrain`, which is set to `False` by default to ensure that the fitting scheme, model selection, hyperparameters, and training are written only once. For details about `retrain`, see Retraining the Model and Exploring Hyperparameters on page 44. This argument will be omitted from discussion and examples until its formal introduction.

In this framework, all outputs share the same set of transformers and machine-learning model. The model set is selected by:

```
obj.set_training_method(target=All, retrain=<b>, parameter_scaler=<k>,
                        target_scaler=<k>, model=<k>)
```

The transformers and models are realized by the scikit-learn package [1] with native hyperparameter options. The settings of the hyperparameters for the transformers and the model are passed by:

```
obj.set_parameter_scaler_hyperparams(target=All, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)

obj.set_target_scaler_hyperparams(target=All, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)

obj.set_model_hyperparams(target=All, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)
```

For example, you can select neural network regression (`model=MLPR`). This model has hyperparameters such as `hidden_layer_sizes` and `activation`, each with default values [3].

You can change the values of some hyperparameters by specifying, for example:

```
obj.set_model_hyperparams(target=All, hidden_layer_sizes=(32, 16, 4),
                          activation="tanh", learning_rate="adaptive")
```

Many-to-One Framework

The many-to-one framework is the default choice. It can also be activated by:

```
obj.set_training_framework(framework=Mx1, retrain=<b>)
```

In this framework, each output has its own set of transformers and machine-learning model. The model set is selected for a single target by:

```
obj.set_training_method(target=<s>, retrain=<b>, parameter_scaler=<k>,
                        target_scaler=<k>, model=<k>)
```

NOTE For the many-to-one framework, the target `All` is still valid and is used to apply the same setting for all targets. The target `All` can serve as an initialization method before detailed settings. For target-specific settings, the arguments `parameter_scaler`, `target_scaler`, and `model` can be omitted if they are not changed from target `All` settings. For both the many-to-many and many-to-one frameworks, `target` takes `All` by default if not provided.

Transformers and models are realized by the scikit-learn package [1] with native hyperparameter options. The setting of the hyperparameters for the transformers and model are passed by:

```
obj.set_parameter_scaler_hyperparams(target=<s>, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)

obj.set_target_scaler_hyperparams(target=<s>, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)

obj.set_model_hyperparams(target=<s>, retrain=<b>,
   <opt_1>=<value_1>,
   <opt_2>=<value_2>,
   ...)
```

For example, you can select neural network regression (`model=MLPR`) for a target named `tar_1`. This model has hyperparameters such as `hidden_layer_sizes` and `activation`, each with default values [3].

You can change the values of some hyperparameters by specifying, for example:

```
obj.set_model_hyperparams(target="tar_1", hidden_layer_sizes=(32, 16, 4),
                          activation="tanh", learning_rate="adaptive")
```

Custom Machine-Learning Model

In addition to the models provided by the SCWML module, labeled by the keywords, you can pass a customized model as the main machine-learning model, provided that it has methods and interfaces conforming to the scikit-learn syntax:

- A fit method takes two NumPy arrays, representing the feature matrix and the output matrix, with the first dimension being the number of samples, and the second dimension being the number of features or outputs. The return value for the fit method is optional.
- A predict method takes one NumPy array, representing the feature matrix, with the first dimension being the number of samples. Its return value is a NumPy matrix containing the predicted target values, with the first dimension being the number of samples.

Data Source: From SCWML Database or External File

The SCWML module retrieves data from the database automatically by default. Before training, the SCWML module checks whether all the results of the submitted experiments are available in the database. If not, then the command queries the database repeatedly until all the data is ready.

Alternatively, you can load training data from an external CSV file, so that the data generation and training of the machine-learning model can be performed separately. The command to load data from a CSV file is:

```
obj.load_data(file=<s>, columns=<t>, rows=<t>)
```

where:

- `file` is the path and name of the data file.
- `columns` is a list of column-renaming rules, in the form of `[<column_name_in_file>, <parameter/target_name_in_ML_module>]`.
- `rows` is a list of row-renaming rules, in the form of `[<row_name_in_file>, <parameter/target_name_in_ML_module>]`.

NOTE The arguments `columns` and `rows` are exclusive. Having the argument `columns` suggests that the file is organized column-wise, and having the argument `rows` suggests that the file is organized row-wise.

Scalar Data Preprocessing: Trimming

To produce a machine-learning model that focuses more on a subset of the data, you can trim the generated dataset to a specific parameter or target range. To add trimming criteria after data generation, specify:

```
obj.add_data_trimmer(name=<s>, range=<l>, abs_value_range=<l>, query=<s>)
```

where:

- `name` is the name of a parameter or target to which to add trimming criteria.
- `range=[<low>, <high>]` controls the range of the passing value. One of the bounds can be `None`.
- `abs_value_range=[<low>, <high>]` controls the absolute range of the passing value. One of the bounds can be `None`.
- `query` is an alternative way to specify more flexible trimming rules. It passes a pandas query clause string to the data trimmer.

Curve Target Preprocessing: Encoding

In general cases, curve data from different simulations might not share the same size of data points or the same range. To fit them into a single machine-learning training and prediction scheme, preprocessing is often necessary. In addition to format considerations, preprocessing also contributes to the quality of a machine-learning model by extracting curve features or by reducing the signal bandwidth. The preprocessing of curve targets is performed by using encoders in the SCWML module.

The point sampling encoder converts curve data into a certain number of data points. The encoder is set by:

```
from scw.ml import PointSamplingEncoder

obj.set_target_encoder(name=<s>,
   encoder=PointSamplingEncoder(num_data_points=<i>, use_floating_range=<b>))
```

where:

- `num_data_points` is the number of data points to sample from each curve.
- `use_floating_range` specifies whether to clip off leading and trailing zeroes for the curves, potentially making the effective range of various curves different. By default, `use_floating_range=False`.

Training the Model

When the necessary model definitions are ready, the machine-learning model can be trained with:

```
obj.train(dataset=<k>, target=<l>|<s>|All, show_plot=<b>, retrain=<b>,
          validation_fraction=<f>, random_seed=<i>)
```

where:

- `dataset` specifies the subset of data on which the model will be trained. Options are:
  - `Full`: Entire dataset
  - `Passed`: Subset that passes the target criteria before data generation
  - `Trimmed`: Subset that passes the data trimming criteria after data generation
  - `PassedTrimmed`: Subset that passes both target criteria and data trimming criteria
- `target` allows you to select one, many, or all targets for the model to train on. For the many-to-many framework, the model can be trained only on all targets at the same time. During training, the entire dataset is split into a training set that is exposed to the model, and a validation set that is blind to the model. After the training, the validation set is checked against the model to evaluate overfitting.
- `show_plot` specifies whether to generate comparison plots of model predictions versus real values for visualization.
- `validation_fraction` specifies the fraction of the validation set out of the dataset (default: 0.1).
- `random_seed` sets the random seed for validating the set selection (default: 1).

After a model is trained, you can reexamine the predictions versus true value plots by using:

```
obj.plot_comparison(data_set=<k>, target=<l>|<s>|All)
```

For visualization purposes, regardless of the training scheme, you can choose any number of targets to visualize by using the argument `target`, provided that the target has been trained on.

Retraining the Model and Exploring Hyperparameters

In a typical machine-learning application, you often want to fine-tune the hyperparameters of a model to obtain the required fitting quality. This is realized by iterations of fitting, evaluation, and adjustments. By default, the SCWML module write-protects a trained model and settings to avoid accidental overwriting. This is achieved by setting `retrain=False` (default). To adjust model settings, you need to activate retraining of a model by specifying `retrain=True`.

NOTE The setting sequence of the machine-learning model is Framework (mxn or mx1) > Model Selection > Hyperparameters > Training. Resetting one step automatically invalidates all of the subsequent steps.

Hyperparameter exploration must be performed manually.

Example: First-Time Training and Retraining

This example demonstrates first-time training followed by evaluation, adjustment, and retraining.

First-time training is performed using the many-to-one framework, with standard normalization and the neural network model:

```
... # Experiment definition, data generation, and so on
obj.set_training_framework(Mx1)
obj.set_training_method(target=All, parameter_scaler=Standard,
   target_scaler=Standard, model=MLPR)
obj.set_model_hyperparams(target=All, hidden_layer_sizes=(32, 16))
obj.train(data_set=Full, target=All, show_plot=True)
obj.save("first_trained")
```

After training, the plots show satisfactory fittings for most targets, but the fitting for parameter `BottomZ` is particularly poor. You decide to check whether changing the width of the neural network will enhance the fitting. This can be achieved in a separate script:

```
import scw.ml.scwml as scwml
obj=scwml.load("first_trained.pkl")
obj.set_model_hyperparams(target="BottomZ", retrain=True,
   hidden_layer_sizes=(64,32))
obj.train(data_set=Full, target="BottomZ", show_plot=True, retrain=True)
obj.save("wider_for_BottomZ")
```

It might take more than one iteration to find an acceptable combination of hyperparameters.

Inference

A trained machine-learning object has the inference capability to predict the output values based on new parameter and condition values. The prediction method is used by specifying:

```
obj.predict(target=<t>, extract=<b>, boundary_check_err=<b>,
            parameters=[("<para_1>", <val_1>), ("<para_2>", <val_2>), ...],
            conditions=[("<cond_1>", <val_1>), ("<cond_2>", <val_2>), ...])
```

The argument `target` specifies the targets to predict and takes `All` by default.

The argument `extract` specifies whether the machine-learning object will return the predictions and update Sentaurus Workbench variables. By default, `extract=True`. If it is set to `False`, then the SCWML module redirects the `DOE: <variable> <value>` outputs to the `predict.out` file in the node directory. The variable relay mechanism allows you to use the output from the machine-learning object as input for subsequent tools.

Prior to the prediction, you can perform a range check on the parameter and condition values. The range check ensures that all the parameter or condition values fall between the extreme values in the training set. The range check is important as it evaluates the suitability of a model on the input. The argument `boundary_check_err` controls the behavior of the command if the check fails:

- If `boundary_check_err=True` (default), then the command throws an exception if the check fails.
- If `boundary_check_err=False`, then the command issues a warning message if the check fails.

The predict function returns three pandas data frames:

- The first `DataFrame` contains the scalar output values as columns and the sample indices as rows.
- The second `DataFrame` contains the metadata for each scalar output, for example, the standard error for a GPR prediction. If no metadata is generated during the prediction process, then each cell contains an empty list.
- The third `DataFrame` contains the prediction results for high-dimensional targets. Each column represents a high-dimensional target, and each row represents a prediction sample. The cell contains a NumPy array with the target values from each dimension.

Model Persistence

The SCWML object can be saved as a pickle (`.pkl`) file by specifying:

```
obj.save("<path/name>", save_base_only=<b>)
```

The created pickle file can be loaded using the `load()` command provided by the SCWML module:

```
from scw.ml import load

obj=load("<path/name>")
```

Instead of saving an entire trained machine-learning object, you can save it as a base object with only the input–output settings, the machine-learning model, and the prediction method for reuse (omitting project management, database input and output, training, and so on):

```
obj.save("<path/name>", save_base_only=True)
```

The argument `save_base_only=False` by default if it is not provided.

Reverse Search

A trained machine-learning model can be used for calibration, which searches for sets of parameters that can produce target output values. A `ReverseSearch` object is used for calibration. The object loads a saved trained machine-learning object with trained machine-learning models. You provide the `ReverseSearch` object with condition values and corresponding target values. The object performs iterations of searches and provides the best candidates that minimize the difference between output metrics and their target values.

To build a `ReverseSearch` object based on a trained machine-learning object, the reverse search script starts with:

```
from scw.ml import ReverseSearch

obj = ReverseSearch(pkl_name=<s>, random_seed=<i>)
```

The argument `pkl_name` specifies the pickle (`.pkl`) file containing the trained machine-learning object, and the `random_seed` is used for the reverse search.

The condition splits and target values corresponding to the condition splits are provided by:

```
obj.set_condition(condition=<s>, values=<f>|<i>|<s>|<l>)

obj.set_target(target=<s>, value=<f>|<s>|<i>, [conditions=<l>])
obj.set_target(target=<s>, file=<s>, variables=<t>, [conditions=<l>])
```

Conditions can be a single value, representing a constant input. They can also be a list of values, representing condition splits. In an experiment, single values and lists with the same size can coexist.

For high-dimensional targets, instead of the target values, the file containing the data and the axis variable names are provided by `file` and `variables`, respectively.

For example, consider a model with `temperature`, `time`, and `type` as inputs, and `BottomZ` as output. In this reverse search, `type` takes a constant value of `"Oxide"`, and the condition splits and target values are:

BottomZ		**time**	
		10	**30**
temperature	300	–1.5	–3.0
	400	–2.0	NA

The constants, condition splits, and target values are registered by:

```
obj.set_condition(condition="type", values="Oxide")

obj.set_condition(condition="time", values=[10, 30])

obj.set_condition(condition="temperature", values=[300, 400])

obj.set_target(target="BottomZ", value=-1.5,
               conditions=[("time", 10), ("temperature", 300)])
obj.set_target(target="BottomZ", value=-2.0,
               conditions=[("time", 10), ("temperature", 400)])
obj.set_target(target="BottomZ", value=-3.0,
               conditions=[("time", 30), ("temperature", 300)])
```

The search criterion is the sum of the weighted square error between the scaled output and the scaled target. By default, each target has a weight of 1.0. The `set_target_weight()` command is used:

```
obj.set_target_weight(target=<s>, weight=<f>)
```

The reverse search is controlled by:

```
obj.reverse_search(num_iterations=<i>, [num_candidates=<i>,] [doe=<s>,]
                   [optimize=<b>])
```

where:

- `num_iterations` sets the number of search iterations to perform and, optionally, `num_candidates` sets the number of parameter set candidates to return (default: 1).
- `doe` sets the sampling strategy to use: `"random"` (default), `"latin_hypercube"`, or `"sobol"`.
- `optimize` specifies whether to perform a gradient-based local calibration for the best candidates after the reverse search (default: `False`).

The command `reverse_search()` returns a list with `num_candidates` of parameter sets ranked by the search criterion from low to high. To retrieve this list after the reverse search, use:

```
obj.get_search_result()
```

Example: Search for High Peak-Gain HBT Device Geometries

This example studies the DC current–voltage behavior of a heterojunction bipolar transistor (HBT) with respect to the device geometries. One simulation flow consists of three steps:

Tool name (tool label)	Description	Command file	Preference file
Sentaurus Structure Editor (`sde`)	Generates device structure	`sde_dvs.cmd`	`sde_dvs.prf`
Sentaurus Device (`Gummel`)	Simulates device	`Gummel_des.cmd`	`Gummel_des.prf`
Sentaurus Visual (`PlotGummel`)	Generates Gummel plot and extracts measurements	`PlotGummel_vis.cmd`	`PlotGummel_vis.prf`

The input parameters include device geometries (width and thickness, shown as arrows in Figure 6), and the doping levels at the emitter and base. There are 15 parameters in total (13 geometric parameters and two doping parameters). The device can be either npn type or pnp type. In this example, the type takes npn as a constant.

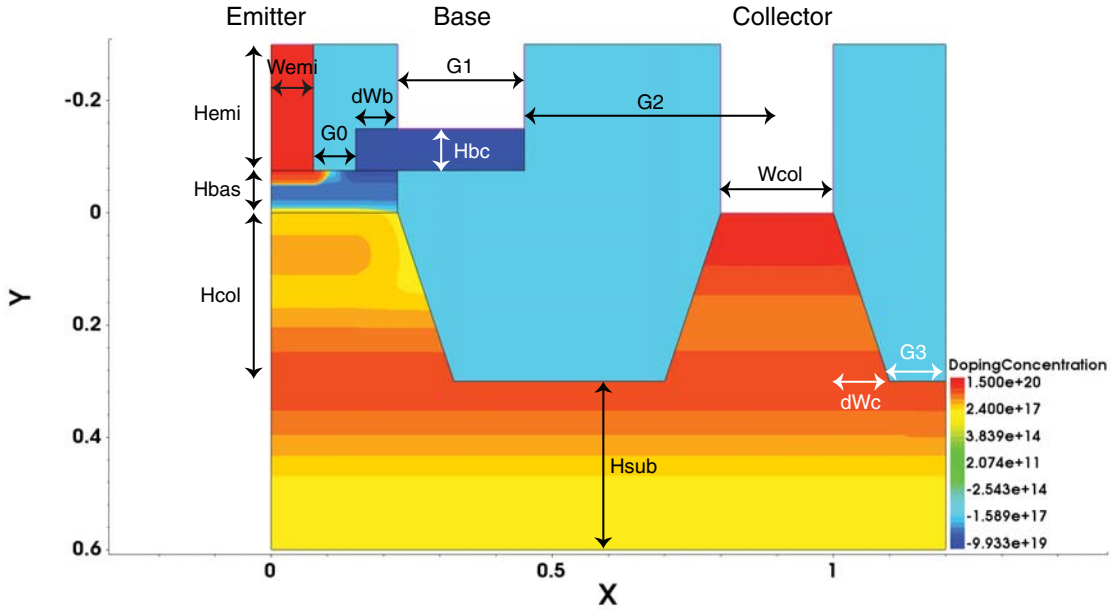

Figure 6 Device structure with geometric parameters labeled

Three outputs are extracted from the simulation. The DC Gummel plot includes the change of the base current, the collector current with respect to the base–emitter voltage, and the ratio between the collector current and the base current as the gain (see Figure 7). The voltage and gain at the peak of the gain–voltage curve is extracted as PeakGain and Vb_PeakGain.

The collector current at Vb_PeakGain is extracted as Ic_PeakGain. The three extracted values are returned in the Sentaurus Visual step using the `puts "DOE: $name $value"` mechanism.

Usually, the PeakGain is of the order of magnitude of hundreds. The question this example will solve is whether it is possible to have an HBT device structure with a peak gain above 1000, while Ic_PeakGain is low (below 1e-5 A/µm).

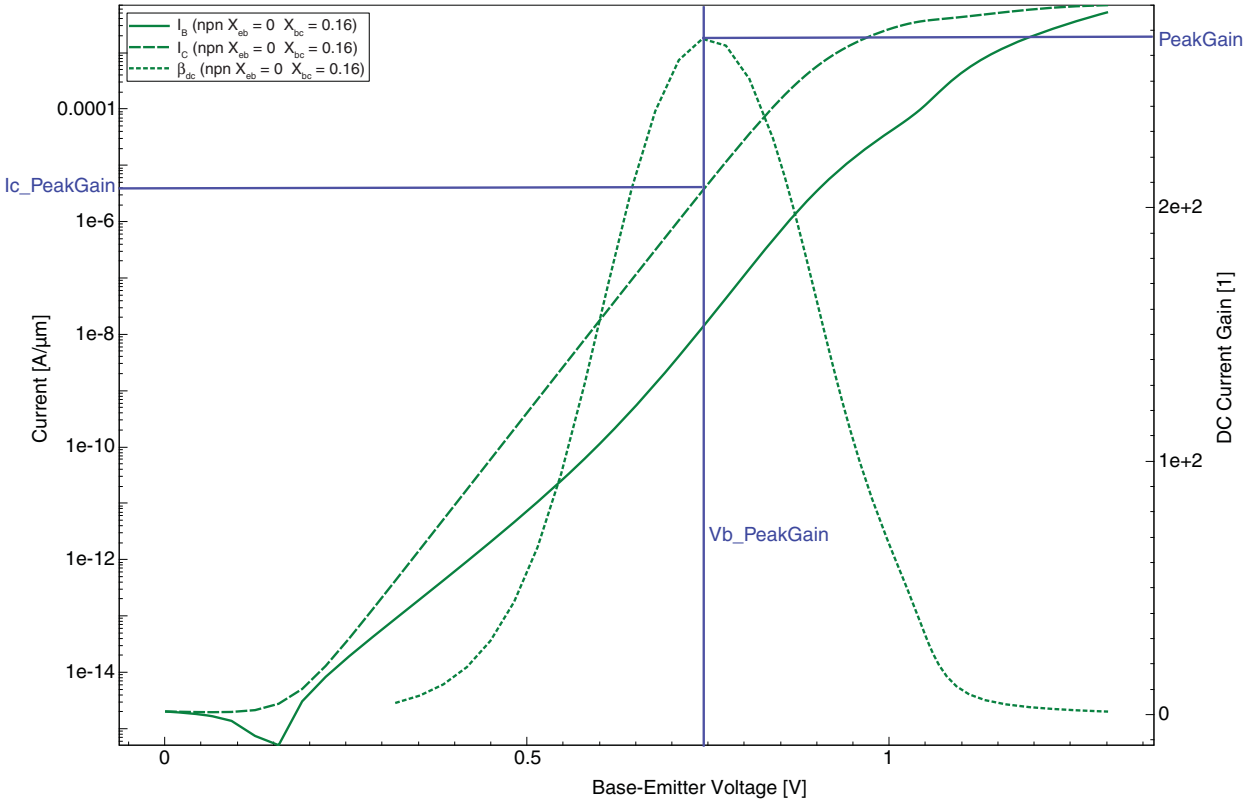

Figure 7 Gummel plot showing extracted targets in example

Tool Setup and Adaptive Data Generation

The tool setup and data generation script is shown here:

```
# Import libraries
from scw.ml import SCWML
from scw.core.parameter import Parameter
```

```
from scw.core.target import Target
from scw.core.scwglobal import *

# Definition of parameters and targets
g0 = Parameter('G0', 0.075, minimum=0.0375, maximum=0.1125)
g1 = Parameter('G1', 0.225, minimum=0.1125, maximum=0.3375)
g2 = Parameter('G2', 0.45, minimum=0.225, maximum=0.675)
g3 = Parameter('G3', 0.1, minimum=0.05, maximum=0.2)
wcol = Parameter('Wcol', 0.2, minimum=0.1, maximum=0.3)
wemi = Parameter('Wemi', 0.15, minimum=0.075, maximum=0.225)
dwc = Parameter('dWc', 0.1, minimum=0.05, maximum=0.15)
dwb = Parameter('dWb', 0.075, minimum=0.0375, maximum=0.1125)
xeb = Parameter('Xeb', 0, minimum=0, maximum=0.32)
xbc = Parameter('Xbc', 0.16, minimum=0, maximum=0.32)
hsub = Parameter('Hsub', 0.3, minimum=0.15, maximum=0.45)
hcol = Parameter('Hcol', 0.3, minimum=0.15, maximum=0.45)
hbas = Parameter('Hbas', 0.075, minimum=0.0375, maximum=0.1125)
hbc = Parameter('Hbc', 0.075, minimum=0.0375, maximum=0.1125)
hemi = Parameter('Hemi', 0.225, minimum=0.1125, maximum=0.3375)

pg = Target('PeakGain', abs_value_range=[750, None])
ic = Target('Ic_PeakGain', abs_value_range=[None, 1.5e-05])
vb = Target('Vb_PeakGain', range=[0.6, 0.8])
dip = Target('HasIbDip', range=[None, 0.5])

# Initialize the machine-learning object
obj = SCWML("@pwd@", db_name="sqdb", parameters=[g0, g1, g2, g3, wcol, wemi,
dwc, dwb, xeb, xbc, hsub, hcol, hbas, hbc, hemi], targets=[pg, ic, vb, dip])

# Set up the child project
obj.set_child(
   child='@pwd@/../internal-HBT',
   queue="sge:tcad",
   tools=["sde", "Gummel", "PlotGummel"],
   files_to_copy=['@pwd@/IN']
)

# Choose the adaptive data generation method
obj.set_data_generation(
   doe="adaptive",
   num_samples=100,
   random_seed=1,
   batch_size=50,
   pass_rate=.9,
   threshold_num_pass=10,
   transformer=Standard,
   classifier=DecisionTree
)
```

```
# The transformer and classifier take default settings. If the settings
# need to change, uncomment the following lines and put in model settings:
# obj.set_adaptive_scaler_hyperparams()
# obj.set_adaptive_clf_hyperparams()

# Create the child project
obj.create_child()

# Save the object before data generation
obj.save("untrained.pkl")

# Begin submitting jobs for data generation
obj.generate_data()
```

Sentaurus Calibration Workbench will finish executing this script after more than 100 data points satisfying the target criteria are generated.

Training the Machine-Learning Model

The machine-learning model setup and training script is shown here:

```
# Load libraries
from scw.ml import load

# Load trained machine-learning object from .pkl file
obj = load("untrained.pkl")

# Set training scheme, machine-learning models, and their hyperparameters
obj.set_training_framework(Mx1)
obj.set_training_method(
    parameter_scaler = Transformer.Standard,
    target_scaler = Transformer.Standard,
    model=Regressor.MLP
)
obj.set_model_hyperparams(max_iter=10000, hidden_layer_sizes=(512,256,128,64),
   tol=1e-4, retrain=True)
obj.set_model_hyperparams(target="PeakGain", max_iter=10000,
   hidden_layer_sizes=(512,256,128,64,32), tol=5e-4, alpha=1e-3, retrain=True)
obj.set_model_hyperparams(target="Ic_PeakGain", max_iter=10000,
   hidden_layer_sizes=(512,256,128,64,32, 16), tol=5e-4, alpha=1e-2,
   retrain=True)

# Trim the dataset for better focus
obj.add_data_trimmer("PeakGain", range=[None, 10000])
obj.add_data_trimmer("Ic_PeakGain", range=[None, 4e-5])
obj.add_data_trimmer("Vb_PeakGain", range=[0.5, 0.8])
```

```
# Train model
obj.train(data_set=Trimmed, target=All, show_plot=False,
validation_fraction=0.1)

obj.save("trained.pkl")
```

After the script is run, a pickle file containing the trained machine-learning model is generated.

Reverse Search

The following script loads the trained model and performs a reverse search for a candidate structure with peak gain of 2000 and a relatively lower collector current (1e-6 A/μm):

```
# Import libraries
from scw.ml import ReverseSearch

# Load machine-learning model to initialize the search engine
obj=ReverseSearch("trained.pkl", random_seed=42)

# Set fixed conditions
obj.set_condition('Type', 'npn')

# Set target values
obj.set_target('PeakGain', 2000)
obj.set_target('Ic_PeakGain', 1e-6)

# Perform reverse search, and return the best candidate among 100000
# candidates
obj.reverse_search(100000)

print(obj.get_search_result())
```

After the script is executed, a parameter name–value dictionary is printed, which produces the best candidate that meets the target values.

Example: Fitting for Etching Profile From Data File

This example demonstrates how to load data from a file, apply name-merging rules and encoding methods for a high-dimensional profile, and perform training, prediction, and reverse search.

The data is prepared before the machine-learning practice and has been extracted into `data.csv` files. The features include:

- Gas_flow_sccm
- C4H8_ratio
- O2_ratio
- Pressure_mTorr
- Power_W
- AC_current_A
- Bias_frequency_MHz
- time_01
- time_02
- time_03

Three profile files are output at each time step:

- profile_01
- profile_02
- profile_03

The profile files are CSV files containing the Z and AverageRadius columns, which define the profile.

During the training, the three time steps and profiles are merged, so that the machine-learning model can predict the resulting profile with the given experiment conditions and time.

Loading Data and Training

```
# Import libraries
from scw.ml import SCWML, PointSamplingEncoder

# Define parameters and targets
feat_names = ['Gas_flow_sccm', 'C4F8_ratio', 'O2_ratio', 'Pressure_mTorr',
'Power_W', 'AC_current_A', 'Bias_frequency_MHz', 'time', 'time_01', 'time_02',
'time_03']
target_names = ['profile', 'profile_01', 'profile_02', 'profile_03']

pars = []
tars = []

for feat in feat_names:
   pars.append(Parameter(feat, 0)) # the value 0 is a placeholder
for tar in target_names:
```

```
   tars.append(Target(tar, simulation_variables=['Z', 'AverageRadius']))

# Initialize SCWML object
obj = SCWML(db_name="sqdb", parameters=pars, targets=tars)

# Set name-merging rules
merge_table = {"time":["time_01", "time_02", "time_03"],
               "profile":["profile_01", "profile_02", "profile_03"]}
obj.merge_feature_target_names(merge_table)

# Set encoding method for merged profile
obj.set_target_encoder("profile",
            PointSamplingEncoder(use_floating_range=True, num_data_points=50))

# Load data from the CSV file
obj.load_data("data.csv")

# Set training configurations and train the model
obj.set_training_framework(Mx1, retrain=False)
obj.set_training_method(target=All, parameter_scaler=Standard,
                        target_scaler=Standard, model=MLPR, retrain=False)
obj.set_model_hyperparams(target=All, retrain=False, max_iter=10000,
                          hidden_layer_sizes=(256,256,256),
                          early_stopping=True, n_iter_no_change=1000,
                          alpha=0, learning_rate='adaptive')
obj.train(target=All, show_plot=False, validation_fraction=0.1, retrain=False,
          random_seed=1)

# Save the model for further use
obj.save("trained.pkl", save_base_only=True)
```

Reverse Search for a Target Profile From a File

The target profile is contained in a `target.csv` file, with the same column names Z and AverageRadius.

```
from scw.ml import ReverseSearch

# Initialize ReverseSearch object with saved pkl file
obj = ReverseSearch("trained.pkl", random_seed=42)

# Set target for profile
obj.set_target("profile", file="target.csv", variables=['Z','AverageRadius'])

# Begin reverse search
obj.reverse_search(100000, doe="latin_hypercube", num_candidates=5,
                   optimize=False)
```

```
obj.reverse_search(100000, doe="random", num_candidates=5,
                   enable_parameter_filter=False, optimize=True)

# Evaluate the reverse search result
import pandas as pd

real_profile = pd.read_csv("target.csv")

parameter_list = [(k, obj.best_param[min(obj.best_param)][k]) for k in
obj.best_param[min(obj.best_param)]]

# Predict returns scalar predictions, metadata for scalar predictions,
# and higher-dimensional predictions
_, _, hdprediction = obj.ml.predict(target="profile", parameters=param_list)

# Plot the profile comparison
import matplotlib.pyplot as plt

plt.figure()
plt.plot(real_profile['AverageRadius'], real_profile['Z'])
plt.plot(hdprediction.loc[0, "profile"][1], hdprediction.loc[0, "profile"][0])

plt.savefig("profile_comparison.png")
```

References

[1] For more information about the scikit-learn package, go to https://scikit-learn.org/stable/index.html.

[2] For more information about the neural network classifier, go to https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

[3] For more information about the neural network regressor, go to https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

CHAPTER 4 Modules for Calibrating Device Simulations

This chapter describes the application-specific calibration modules for device simulations.

This chapter presents the calibration modules for devices:

- General Device Calibration: devcal Module
- Hot-Carrier Stress Degradation Calibration

General Device Calibration: devcal Module

The `devcal` module allows you to calibrate device simulators by defining your own calibration strategy as well as application-specific predefined calibration strategies.

At the beginning of the Sentaurus Calibration Workbench command file, import the module. For example:

```
from scw.apps.devcal import DevCal
```

Then, instantiate a calibration object from the imported module class. For example:

```
devcal = DevCal()
```

Now, this calibration object can be used to manage the calibration flow, including parameter, target, option, and task specifications (see Chapter 2 on page 5).

Hot-Carrier Stress Degradation Calibration

With the trend toward technology miniaturization and the need for higher on-chip integration densities, device reliability issues are a major concern. Leakage and reliability are especially critical when designing input and output (IO) devices or memory peripheral devices, which operate at high voltage. The charged carriers in the channel of such transistors, accelerated by the large electric field, can gain enough high energy to damage the semiconductor–insulator interface.

The term *hot-carrier degradation* in modern devices comprises several microscopic physical mechanisms, such as interface Si-H bond dissociation by interaction with single high energetic (*hot*) carriers or with a series of *cold* carriers, bond breakage due to field-enhanced thermal interaction with the lattice, and the impact of bond energy dispersion on interface traps generation.

The hot-carrier stress (HCS) degradation model in Sentaurus Device combines all these microscopic physical aspects into a simple analytic description, suitable for modeling time-dependent interface trap generation. For details, see *Sentaurus™ Device User Guide*, Hot-Carrier Stress Degradation Model.

However, the HCS degradation model, based on [1], was originally designed for IO devices, more specifically for lateral double-diffused metal–oxide–semiconductor (LDMOS) transistors with shallow trench isolation. Even though the essential physics of hot-carrier degradation is accounted for in the HCS degradation model, major adjustments to the model parameters might be needed, when applying it to the degradation study of a different technology. Such a parameter calibration process can be complex and time-consuming, since it requires a deep understanding of the models involved.

The `devcal` module is designed to overcome this limitation, by providing a tool for fast experimental data calibration or, at least, for a significant support to it. In addition to interface trap formation, another important aspect of the hot-carrier degradation analysis consists of modeling the detrimental effect that generated traps have on carrier mobility. Therefore, to provide a complete tool for the calibration of all the I_d–V_g characteristics at different stress times, which can be used to extract the threshold voltage or the drain current variations as a function of stress time, the `devcal` module also takes into account, in the calibration process, the parameters of the Coulomb scattering mobility model that depends on the interface trap concentration as well as the parameters to define the energy distribution of the generated traps. For details, see *Sentaurus™ Device User Guide*, Mobility Degradation Components due to Coulomb Scattering.

The `devcal` module has been tested mostly on measurements of memory peripheral MOSFETs. However, it can help the calibration process of MOS-based devices in general.

Input Data

Two calibration commands are available: one for NMOS devices and one for PMOS devices. The degraded NMOS experimental curves can be calibrated with acceptor-type traps only. For PMOS devices, both donor- and acceptor-type traps are needed. This implies some minor differences in the required inputs to the calibration functions that are presented here.

You must provide the following inputs:

- As part of the parent project, the Sentaurus Device command file, which includes all the commands needed to perform the time-dependent degradation simulation followed by the I_d–V_g simulation post-degradation, must be provided. In detail, quasistationary ramps to the stress biases, followed by the transient degradation simulation with different final times, must be performed. The transient final time must be defined as a Sentaurus Workbench parameter, for example, `time`, and added as a condition parameter for targets in Sentaurus Calibration Workbench. The values correspond to the stress times at which the experimental curves are extracted.

  The HCS degradation model must be activated in the `Traps` statement of the `Physics` section relative to the material or region interface where the traps are generated, with the options `SHE` and `BondDispersion` (see *Sentaurus™ Device User Guide*, Hot-Carrier Stress Degradation Model).

  For NMOS devices, this would be:

```
Traps(
   (Acceptor Gaussian EnergyMid=@EmidA@ EnergySig=@EsigA@
    FromMidBandGap eHCSDegradation(SHE BondDispersion))
)
```

  Both trap types are needed for PMOS devices:

```
Traps (
   (Acceptor Gaussian EnergyMid=@EmidA@ EnergySig=@EsigA@ FromMidBandGap
    eHCSDegradation(SHE BondDispersion))
   (Donor Gaussian EnergyMid=@EmidD@ EnergySig=@EsigD@ FromMidBandGap
    hHCSDegradation(SHE BondDispersion))
)
```

  As shown here, a Gaussian energy distribution is assumed for the generated traps, with the arbitrary initial values of `EnergyMid` (position of the Gaussian peak) and `EnergySig` (standard deviation) defined by Sentaurus Workbench parameters.

  The corresponding calibration parameters must be defined for Sentaurus Calibration Workbench in the following way. For NMOS devices, this is:

```
EmidA = Parameter('EmidA', 0, -0.5, 0.5)
EsigA = Parameter('EsigA', 0.1, 0.1, 0.3)
```

  For PMOS devices, this is:

```
EmidA = Parameter('EmidA', -0.35, -2.0, 0.0)
EsigA = Parameter('EsigA', 0.5, 0.05, 1.0)
EmidD = Parameter('EmidD', -0.55, -0.7,-0.3)
EsigD = Parameter('EsigD', 0.2, 0.1, 0.4)
```

  The Coulomb scattering mobility model that depends on the interface trap concentration must be activated in the semiconductor `Physics` section of the Sentaurus Device input file

for the material or region interface where the traps are generated (see *Sentaurus™ Device User Guide*, Mobility Degradation Components due to Coulomb Scattering).

- In the Sentaurus Calibration Workbench input file, the Sentaurus Device parameter file must also be specified in the `set_option()` command in the list of `files_to_copy`. This file must not include the parameters of the HCS degradation model and the Coulomb scattering mobility model. They are calibrated by the `devcal` calibration module starting from the default values and provided as output of the simulation.
- The `.plt` file contains the gate voltage and drain current data to generate the I_d–V_g characteristics.
- If the degradation simulation and the post-degradation I_d–V_g simulation are run separately, then the `.tdr` file is required, which is used to transfer the generated interface trap distribution.
- The targets must be defined in the following way:

  If you want to directly compare the measured and simulated I_d–V_g characteristics, then the I_d–V_g data must be defined as profile targets. In this case, Sentaurus Calibration Workbench automatically calculates internally and minimizes the root mean square (RMS) error of such curves.

  However, if you want to define a different type of error to be optimized (for example, linear or saturation drain current variation at a specific gate voltage, or threshold voltage variation), then you must provide a Sentaurus Visual script for the error and variable extraction, and define the corresponding scalar targets.

  In either case, for the classification of the provided targets, you must follow a specific target type for I–V plots:

  - Include `type=IdVgLin` in the target definition for linear drain currents.
  - Include `type=IdVgSat` in the target definition for saturation drain currents.
  - Include `type=IdVgOffSat` in the target definition for off-state saturation drain currents.

Calibration Strategies

The calibration strategies are implemented in the `devcal` calibration functions. They consist of a sequence of parameter set evaluations, conditional operations, and gradient-based calibration steps, which follows the logic of calibration work performed by expert engineers.

To develop and test the calibration strategies, experimental degraded I_d–V_g characteristics in linear and saturation regimes at different stress times, different stress bias conditions (on-state and off-state degradation) for both n- and p-type devices at two different temperatures have been used. In general, the experimental data shows not only a threshold voltage shift, but also a significant subthreshold slope degradation and, in some cases, a nonmonotonic current variation with stress time. To reproduce these effects, the parameters of the generated trap

energy distribution and the mobility degradation due to traps must be calibrated, in addition to the HCS degradation model parameters.

Due to its complexity and the large number of calibration parameters, the calibration task is divided into subtasks, which are performed in a specific order. Starting from the physics-limited range of variation for each parameter, the idea is to restrict the variation range with evaluation steps before performing a gradient-based calibration step, thereby increasing the chances of success for the calibration. The calibration strategy exploits the characteristics of the HCS degradation model by allowing the separate analysis of its three interface trap formation contributions, that is, the single-particle process, the multiple-particle process, and the field-enhanced thermal process (see *Sentaurus™ Device User Guide*, Hot-Carrier Stress Degradation Model).

Due to the different behavior shown by the experimental data and the formation of both acceptor and donor traps, the calibration strategy for PMOS devices differs from the one for NMOS devices, and it requires a longer turnaround time.

The `devcal` calibration command for NMOS devices is:

```
devcal.calibrate_hcsd_nmos(parameters=<t>,
                           targets=<s>,
                           use_thermal=<b>,
                           condition_variable=<s>,
                           parent=<s>,
                           child_dir=<s>)
```

The argument `parameters` includes a list of exactly two user-defined Sentaurus Workbench parameters. The first one is the Sentaurus Device acceptor trap distribution central energy `EmidA`, and the second one is the Sentaurus Device acceptor trap distribution energy width `EsigA`. For details about the definition of these parameters, see Input Data on page 58.

The argument `use_thermal` specifies whether to use the field-enhanced thermal component of the HCS degradation model in Sentaurus Device, which is set to `True` by default.

The target condition parameter for the transient final time, for example, `time`, must be specified by the argument `condition_variable`.

For a description of all arguments, see calibrate_hcsd_nmos on page 97.

The `devcal` calibration command for PMOS devices is:

```
devcal.calibrate_hcsd_pmos(parameters=<t>,
                           targets=<s>,
                           condition_variable=<s>,
                           parent=<s>,
                           child_dir=<s>)
```

The argument `parameters` includes a list of exactly four user-defined Sentaurus Workbench parameters:

- The first one is the Sentaurus Device acceptor trap distribution central energy `EmidA`.
- The second one is the Sentaurus Device acceptor trap distribution energy width `EsigA`.
- The third one is the Sentaurus Device donor trap distribution central energy `EmidD`.
- The fourth one is the Sentaurus Device donor trap distribution energy width `EsigD`.

For details about the definition of these parameters, see Input Data on page 58.

The target condition parameter for the transient final time, for example, `time`, must be specified by the argument `condition_variable`.

For a description of all arguments, see calibrate_hcsd_pmos on page 98.

References

[1] S. Reggiani *et al.*, “TCAD Simulation of Hot-Carrier and Thermal Degradation in STI-LDMOS Transistors,” *IEEE Transactions on Electron Devices*, vol. 60, no. 2, pp. 691–698, 2013.

[2] G. Torrente *et al.*, “Hot Carrier Stress modeling: from degradation kinetics to trap distribution evolution,” in *IEEE International Integrated Reliability Workshop (IIRW)*, South Lake Tahoe, CA, USA, pp. 134–137, October 2015.

CHAPTER 5 Modules for Calibrating Power Device Simulations

This chapter describes the application-specific calibration modules for power device simulations.

Sentaurus Device is an essential simulator for the development of power electronics technology. You can use Sentaurus Calibration Workbench to calibrate the parameters of power device models in AC and DC characteristics.

The `igbtcal` module provides calibration strategies for insulated gate bipolar transistor (IGBT) AC and DC characteristics.

This chapter describes the calibration module for power devices:

- IGBT Device Calibration: igbtcal Module

IGBT Device Calibration: igbtcal Module

The insulated gate bipolar transistor (IGBT) is one of the key components in various power applications due to its low conductivity and switching losses. It is introduced to take advantage of both power MOSFET and BJT devices to achieve optimal device characteristics. Typically, the Sentaurus Process and Sentaurus Device simulators are used to characterize the optimal conditions for achieving the required performance of an IGBT. However, the accuracy of the simulation depends on experience, knowledge of model selection, and parameter calibration for the appropriate models. The selection of a physical model according to each device characteristic and a well-defined calibration strategy is essential for the success of IGBT calibrations.

The `igbtcal` module is designed to guide the calibration strategy of model parameters for IGBT device characteristics using Sentaurus Device. This module provides two calibration strategies: DC and AC characteristics. The DC calibration strategy optimizes the model parameters by calibrating the I_c–V_g, I_c–V_c, and breakdown voltage data automatically. In particular, I_c–V_g and I_c–V_c data can be performed in a two-step calibration, which is useful to calibrate temperature-dependent parameters. The AC calibration strategy includes steps to calibrate the model and parameters to fit C–V, gate charging (Q_g), and switching characteristics. Here, the measured data (target) should be curve or point data to be compared with the calibrated simulation results. In addition, it is important to select the parameters most

sensitive to the target characteristics. As the number of parameters increases, the number of iterations increases.

At the beginning of the Sentaurus Calibration Workbench command file, import the module. For example:

```
from scw.apps.igbtcal import IGBTCal
```

Then, instantiate a calibration object from the imported module class. For example:

```
igbtcal=IGBTCal()
```

Now, this calibration object can be used to manage calibration inputs including parameter, target, option, and task specifications (see Chapter 2 on page 5).

`Target` and `Parameter` for the `igbtcal` module require you to specify the type of the simulation data, so that the module can classify the parameters and targets for each calibration step:

```
Parameter("C", 5.8e2, 5.8e1, 5.8e3, Log10, type=IcVg)

Target("sIcVg", id="IcVg", value="IGBT_25C_IcVg.plt",
       conditions={"IcVg": "1", "IcVc": "0", "BV":"0"},
       reference_variables=["x", "y"],
       simulation_variables=["gate OuterVoltage", "collector TotalCurrent"],
       error_method="log_abs", x_range=[0.5, 15.0],
       type=IcVg
)
```

The available types are as follows:

- DC calibration:
  - IcVg
  - IcVg2
  - IcVc
  - IcVg2
  - BV
- AC calibration:
  - CV
  - Qg
  - Switching

In `Target`, if the `error_method` is not specified, then the default error method is applied based on each type of simulation. For details, see Target on page 129.

IGBT DC Characteristics Calibration

The IGBT calibration module `igbtcal` allows you to calibrate IGBT DC device characteristics – I_c–V_g, I_c–V_c, and breakdown voltage – using interface charge and mobility, resist and high-field saturation, V_t shift, breakdown voltage, and so on.

To run an evaluation task for all included targets with the current parameter values, specify:

```
igbtcal.evaluate_dc(parameters, targets)
```

The calibration strategy performs multi-step calibration for the targets and parameters classified by types: IcVg, IcVc, and BV. In particular, IcVg and IcVc types can be run in a two-step calibration if targets and parameters specified by `type=IcVg2` or `type=IcVc2` are included. The BV calibration step performs the design-of-experiments (DoE) search first, then it starts the calibration at parameters with the minimum RMS error obtained from the initial search. If targets are not found for a specific type, then the calibration step for that type is omitted. In addition, the target tolerance (one of the calibration stopping criteria) for each step can be controlled separately.

To increase accuracy, some parameters are recommended to include for calibration: *fixed charge* at the silicon–oxide interface (`Qit`) for IcVg and *contact resistance* (`resist`) for IcVc. In particular, if the `igbtcal` module cannot find a parameter named `resist` (case insensitive) in the IcVc parameter list, then the module omits the contact resistance calibration step.

The following command executes the DC calibration strategy:

```
igbtcal.calibrate_dc(parameters=<t>,
                     targets=<t>,
                     [parent=<s>,]
                     [child_dir=<s>,]
                     [tolerance_icvg=<f>,]
                     [tolerance_icvc=<f>,]
                     [tolerance_bv=<f>,]
                     [max_iteration=<i>,]
                     [doe_method=<s>,]
                     [doe_nsamples=<i>])
```

For a description of the arguments, see calibrate_dc on page 92.

IGBT AC Characteristics Calibration

The IGBT calibration module `igbtcal` allows you to calibrate IGBT AC device characteristics – C–V, gate charge (Q_g), and switching simulations – using parasitic components, lifetime, recombination parameters, and so on.

In the AC calibration strategy, `Parameter` and `Target` are classified as `type=CV`, `Qg`, and `Switching`.

To run an evaluation task for all included targets with the current parameter values, specify:

```
igbtcal.evaluate_ac(parameters, targets)
```

The following command executes the AC calibration strategy:

```
igbtcal.calibrate_ac(parameters=<t>,
                     targets=<t>,
                     [parent=<s>,]
                     [child_dir=<s>,]
                     [tolerance_cv=<f>,]
                     [tolerance_qg=<f>,]
                     [tolerance_switching=<f>,]
                     [max_iteration=<i>])
```

For a description of the arguments, see calibrate_ac on page 91.

CHAPTER 6 Modules for Calibrating Topography Simulations

This chapter describes the application-specific calibration modules for topography simulations.

Sentaurus Topography 3D simulates deposition, etching, and simultaneous etching and deposition processes using physical and geometric models. Two simulation methods are used to implement the physical models: the level set method and the particle Monte Carlo method.

This chapter describes the calibration modules for topography:

- PMC Topography Calibration: pmccal Module
- HARC Etching Calibration

PMC Topography Calibration: pmccal Module

The `pmccal` module allows you to calibrate the particle Monte Carlo (PMC) method by defining your own calibration strategy. It also includes application-specific predefined calibration strategies as well as specific features to take into account the variability of the PMC method.

NOTE For the level set method, the `pmccal` module can be used as well, but so can the Sentaurus Calibration Workbench (`scwcal`) module.

At the beginning of the Sentaurus Calibration Workbench command file, import the module. For example:

```
from scw.apps.pmccal import PMCCal
```

Then, instantiate a calibration object from the imported module class. For example:

```
pmccal = PMCCal()
```

Now, this calibration object can be used to manage the calibration flow, including parameter, target, option, and task specifications (see Chapter 2 on page 5).

Reevaluation due to Variability

PMC and Monte Carlo flux calculations can show numeric variability from run to run due to numeric noise, even for the same parameter set. As a consequence, the evaluation and also the parameter calibration can show variability.

By default, simulations for a specific parameter and target set is only evaluated once, and results are reused in the following tasks if the parameter set and the target set are identical. However, for PMC calibration, it is beneficial to reevaluate it to prevent the calibration converging in a local optimum due to numeric noise and to continue to find a global optimum. Therefore, the evaluation and gradient-based local calibration tasks will reevaluate all simulations by default.

To run a simple evaluation task for all included targets with the current parameter values without reevaluation, use the command:

```
cal.evaluate(parameters=<t>, targets=<t>, reevaluate=False)
```

To run a gradient-based local calibration task for all included targets with the current parameter values without reevaluation, use the command:

```
cal.optimize(parameters=<t>, targets=<t>, reevaluate=False)
```

See evaluate on page 103 and optimize on page 112.

Sentaurus Topography 3D Target Extraction

To extract the evaluated critical dimensions (CDs) of a Sentaurus Topography 3D simulation, the Sentaurus Topography 3D command file should include the extraction part at the end of the topography simulation. There is no need for an external extraction tool such as Sentaurus Visual.

For the PMC method, it is recommended to use the `extract type=shape_analysis` command of Sentaurus Topography 3D, because it provides averaging as well as smoothing, which is beneficial for calibration.

For the extraction of a CD for an etched hole, shape analysis extraction based on a cylinder hole should be used. For example, in the case of the necking CD and position, it is used to extract the z-coordinates, polymer surface coverage, and average radius:

```
set FHL [extract name=Full_Hole \
   type=shape_analysis reference_shape=cylinder_hole output_type={ } \
   point1={0.1 0.1 @structure_bottom@} point2={0.1 0.1 @structure_top@} \
   max_radius=0.1 smoothing_order=25 top_shift=0.005]
```

```
set Z_coords      [lindex [lsearch -regexp -inline [lindex $FHL 1] Z] 1]
set SurfCov_P_Avg [lindex [lsearch -regexp -inline [lindex $FHL 1]
   SurfaceCoverage_P_Avg] 1]
set Radius_Avg [lindex [lsearch -regexp -inline [lindex $FHL 1] Radius_Avg] 1]
```

Based on these quantities, the extraction of the specific CD and its location is performed:

```
set NeckingCD 1.0e5
set NeckingZ  0.0

foreach zcoord $Z_coords surfcover $SurfCov_P_Avg radiusavg $Radius_Avg {
   if {$radiusavg<$NeckingCD && $zcoord>$BowingZ && $radiusavg>0 &&
   $surfcover>0.98} {
      set NeckingCD $radiusavg
      set NeckingZ  $zcoord
   }
}

set NeckingCD [expr 2*$NeckingCD]
```

Finally, variable extraction in the style of Sentaurus Workbench is needed:

```
puts "DOE: NeckingZ  [format %.4g $NeckingZ]"
puts "DOE: NeckingCD [format %.4g $NeckingCD]"
```

For each extracted variable, a scalar target must then be defined in Sentaurus Calibration Workbench (see Scalar Targets on page 9).

HARC Etching Calibration

For the fabrication of DRAM and 3D NAND memory devices, a crucial process step is the formation of high-aspect ratio contact (HARC) holes by plasma etching. To calibrate HARC formation, it is important to study the influence of process parameters on the contact hole shape, including the hole depth and diameter, as well as the necking and bowing of the sidewalls (see Figure 8 on page 72). To utilize the PMC method of Sentaurus Topography 3D for such studies, it is critical to calibrate the corresponding plasma etching models based on specific process parameters and their measured shapes.

The `pmccal` module provides two predefined comprehensive calibration strategies for high-aspect ratio (HAR) etching:

- One based on a gradient-based local calibration algorithm (see Calibration Strategy by Gradient-Based Local Calibration Algorithm on page 70)
- One based on machine learning (see Calibration Strategy by Machine Learning on page 72)

In addition, the calibration module allows you to define your own calibration strategies.

Calibration Strategy by Gradient-Based Local Calibration Algorithm

The predefined calibration strategy for HARC etching, based on the gradient-based local calibration algorithm, includes many calibration tasks, which are grouped in the initial, middle, and final parts. When the total RMS error is within the tolerance or the allowed number of tasks is exceeded, the calibration strategy switches from the initial part to the middle part, and then from the middle part to the final part, which ends the calibration.

The following command executes the calibration strategy:

```
pmccal.calibrate_har_etch(parameters=<t>,
                          targets=<t>,
                          parent=<s>,
                          child_dir=<s>,
                          tolerance_end=<f>,
                          tolerance_mid=<f>,
                          tolerance_init=<f>,
                          max_iteration_end=<i>,
                          max_iteration_mid=<i>,
                          max_iteration_init=<i>,
                          weight_factor=<f>,
                          reevaluate=<b>,
                          use_bi_sensitivity=<b>)
```

For a description of the arguments, see calibrate_har_etch on page 93.

The calibration strategy is based on a classification of calibration parameters and targets. Since Sentaurus Calibration Workbench does not know the Sentaurus Topography 3D API, it does not automatically recognize the role of each model parameter being parameterized or the type of extracted variables. Therefore, the naming scheme of calibration parameters needs to differentiate between different models and materials. In addition, targets and extracted variable names must follow strict naming schemes. See Parameter Naming Schemes and Target Naming Schemes on page 71.

Parameter Naming Schemes

This parameter naming scheme is case insensitive. Use the following naming scheme for correct parameter classification:

- Yield functions: `yield_<type>_<material>`
- Sputtering: `p_sputter_<type>_<material>`

- Reflection: `p_reflect_<type>_<material>`
- Deposition (sticking): `sc_<type>_<material>`
- Redeposition: `sc_redep_<type>_<material>`
- Chemical etching: `rad_<type>_<material>`
- Flux: `flux_<type>`

The `<type>` can be any arbitrary string you have defined.

The supported options for `<material>` are:

- For hard masks: `mask, photoresist, polymer`
- For the bulk: `bulk, nitride, oxide, silicon`

Target Naming Schemes

This target naming scheme is case insensitive. Use the following naming scheme for correct target classification:

```
t_<condition>_<extraction>
```

The string `<condition>` describes the conditions unambiguously.

The supported extracted variables used for the string `<extraction>` are:

- Depth and bottom width of bulk hole: `BottomCD, BottomZ`
- Thickness and top width of remaining hard mask: `RemainingHMCD, RemainingHMZ`
- Width at hard mask–bulk interface position: `HMBottomCD`
- Position and width of smallest opening (by deposition): `NeckingCD, NeckingZ`
- Position and width of largest opening of bulk hole: `BowingCD, BowingZ`

Additional targets can also be used, but the proposed targets are required to allow for a successful calibration result.

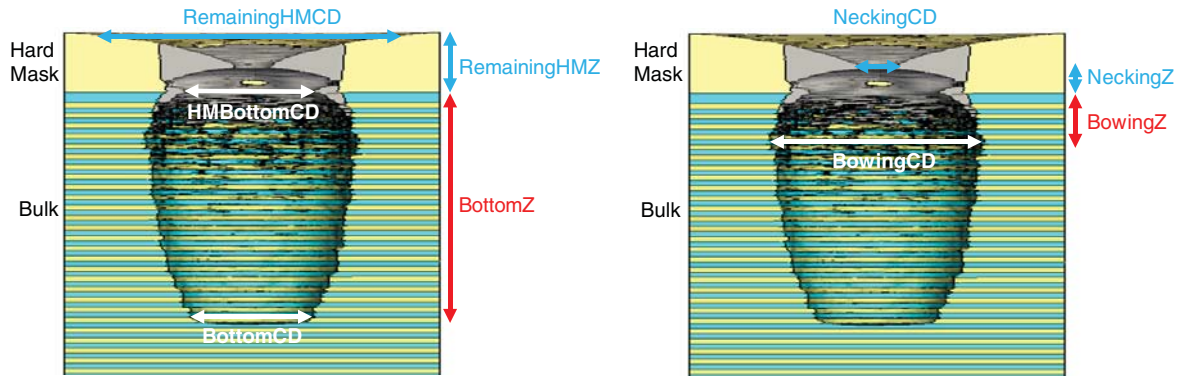

Figure 8 Illustration of extracted variables

Calibration Strategy by Machine Learning

The predefined calibration strategy for HARC etching, based on machine learning, includes parameter sensitivity analysis, training data computation, training the machine-learning model, and finally searching for the best candidates by machine-learning inference. This machine-learning calibration strategy, by default, performs a calibration for user-specified parameters and targets and has been validated for different HARC etching examples. However, you can create user-defined strategies by using a set of machine-learning base functions (see Creating a User-Defined Machine-Learning Strategy on page 75).

Unlike the calibration strategy for gradient-based local calibration, this strategy does not require using a specific naming scheme for calibration parameters, targets, and extracted variables. This means you can assign any arbitrary names to parameters, targets, and extracted variables, because they do not need to be classified. However, the same naming schemes as for the gradient-based local calibration can still be used.

As usual, calibration parameters are specified by `Parameter()` with nominal, minimum, and maximum values, where the minimum and maximum values define the parameter range. The machine-learning strategy task performs a sequence of subtasks (`explore` and `refine`). The outcomes of these subtasks are calibrated parameters and a file where the corresponding machine-learning models are saved.

The `calibrate_har_etch_ml()` command executes the machine learning–based calibration strategy for HARC etching (see calibrate_har_etch_ml on page 94).

For example, a typical use case would be:

```
pmccal.calibrate_har_etch_ml(parameters=parameters, targets=targets,
                             tolerance=0.07, num_parameters=0,
                             parent="../parent/HARC", constraint="Fail==0",
                             child_dir="../ML1")
```

Machine-learning projects are created in the subdirectory `../ML1` as specified by the `child_dir` argument.

The argument `parent` specifies the parent Sentaurus Workbench project.

By specifying `num_parameters`, you can control the number of features used for training. If you specify `num_parameters=0`, then all available parameters are used as features.

The constraint `"Fail==0"` means that only experiments with the Sentaurus Workbench extracted variable `Fail` equal to 0 are selected. Moreover, any Sentaurus Workbench variable such as `"Valid==1"`, as well as more complex expressions such as `"Fail==0 and split==1"`, can be used for `constraint`. Usually, `constraint` is used to discard poor data from machine-learning training.

The calibration strategy performs a sequence of `explore` and `refine` subtasks until either the `tolerance` is reached or the maximum number of internal iterations is reached.

Some additional control is allowed by setting the argument `tolerance_refine`:

- If the weighted RMS error in the current internal iteration is less than `tolerance_refine`, then the `refine` subtask is executed.
- Otherwise, the `explore` subtask is executed.

In addition, you can define target-specific tolerances. The target-specific tolerance should be set to a value that considers the overall global tolerance, the required accuracy for that specific target, and noise levels.

The following example shows how to set tolerances for individual targets:

```
t_BottomZ = Target(name='BottomZ', value=<f>, id='t_BottomZ', tolerance=0.01)
t_RemainingHM = Target(name='RemainingHM', value=<f>, id='t_RemainingHM',
   tolerance=0.05)
t_BowingZ = Target(name='BowingZ', value=<f>, id='t_BowingZ', tolerance=0.1)
t_NeckingZ = Target(name='NeckingZ', value=<f>, id='t_NeckingZ',
   tolerance=0.1)
t_BottomCD = Target(name='BottomCD', value=<f>, id='t_BottomCD',
   tolerance=0.08)
t_RemainingHMCD = Target(name='RemainingHMCD', value=<f>,
   id='t_RemainingHMCD', tolerance=0.08)
```

```
t_BowingCD = Target(name='BowingCD', value=<f>, id='t_BowingCD',
   tolerance=0.05)
t_NeckingCD = Target(name='NeckingCD', value=<f>, id='t_NeckingCD',
   tolerance=0.08)
t_HMBottomCD = Target(name='HMBottomCD', value=<f>, id='t_HMBottomCD',
   tolerance=0.08)
```

In addition to the predefined calibration strategy command `calibrate_har_etch_ml()`, there are other machine-learning base commands for the following applications:

- Machine-Learning Model Exploration
- Machine-Learning Model Refinement

Machine-Learning Model Exploration

The `explore_har_etch_ml()` command executes the exploration of a machine-learning model (see explore_har_etch_ml on page 106). The arguments of this command have the same meaning as those for `calibrate_har_etch_ml()` as described in Calibration Strategy by Machine Learning on page 72.

The calibration task of `explore_har_etch_ml()` corresponds to the `explore` subtask of `calibrate_har_etch_ml()`.

The exploration algorithm works as follows:

1. A child project for machine-learning training is created in the directory `child_dir` and is run.
2. Machine-learning models are created for each target and split condition. A machine-learning model database is populated.
3. The machine-learning database is used to predict calibration candidates. These candidates are simulated, and the best candidate is selected.
4. The exploration is repeated until the maximum number of iterations (`num_iterations`) is reached.

Machine-Learning Model Refinement

The `refine_har_etch_ml()` command executes the refinement of a machine-learning model (see refine_har_etch_ml on page 117). The arguments of this command have the same meaning as those for `calibrate_har_etch_ml()` as described in Calibration Strategy by Machine Learning on page 72.

The calibration task of `refine_har_etch_ml()` corresponds to the `refine` subtask of `calibrate_har_etch_ml()`.

The refinement algorithm works as follows:

1. A child project for machine-learning training is created in the directory `child_dir` and is run.
2. A machine-learning model is trained in proximity of the current nominal parameter values and is used to predict new experiments. These experiments are simulated, and the best experiment is selected.
3. Several iterations are performed until the maximum number of iterations (`num_iterations`) is reached or the convergence criteria are met.

Creating a User-Defined Machine-Learning Strategy

The simplest way to run the predefined strategy is *as is*. In this case, the machine-learning calibration is performed until either the `tolerance` is achieved or the maximum number of internal iterations is reached.

If the achieved calibration is not accurate enough, then you can extend the calibration strategy or even create a user-defined one. For this purpose, you can use the machine-learning base commands for machine-learning model exploration (see Machine-Learning Model Exploration) and machine-learning model refinement (see Machine-Learning Model Refinement).

When combining several calibration strategy and base commands, task-specific names for the machine-learning tasks are required to distinguish the different tasks. Task names are defined by setting the argument `task_name`.

The following example shows how to use the `refine_ml()` command to fine-tune the calibration parameters after a `calibrate_har_etch_ml()` task:

```
pmccal.calibrate_har_etch_ml(parameters=parameters, targets=targets,
                             task_name="PMC_HAR", tolerance=0.07,
                             num_parameters=0, constraint="Fail==0",
                             parent="../parent/HARC", child_dir="../ML1")

pmccal.refine_ml(parameters=parameters, targets=targets,
                 task_name="PMC_HAR_1", tolerance=0.07,
                 num_parameters=0, constraint="Fail==0",
                 parent="../parent/HARC", child_dir="../ML1",
                 sensitivity="PMC_HAR")
```

Here, the machine-learning tasks `PMC_HAR` and `PMC_HAR_1` are executed sequentially. The `PMC_HAR` task performs a calibration with `tolerance=0.07`. The calibrated parameter values from the `PMC_HAR` task are then used as the initial values for the `PMC_HAR_1` task, which performs a `refine` task with target `tolerance=0.07`.

The argument `sensitivity="PMC_HAR"` specifies that the sensitivity analysis from task `PMC_HAR` is used in task `PMC_HAR_1`, reusing existing simulations and reducing the total number of simulations.

The following example shows how to perform a `refine` task after an initial calibration using only a subset of the targets:

```
pmccal.calibrate_har_etch_ml(parameters=parameters, targets=all_targets,
                             task_name="PMC_HAR", tolerance=0.07,
                             num_parameters=0, constraint="Fail==0",
                             parent="../parent/HARC", child_dir="../ML1")

pmccal.refine_ml(parameters=parameters, targets=targets_wo_BottomCD_NeckingCD,
                 task_name="PMC_HAR_1", tolerance=0.07, num_parameters=0,
                 constraint="Fail==0", parent="../parent/HARC",
                 child_dir="../ML1", sensitivity="PMC_HAR")
```

The task `PMC_HAR_1` performs a `refine` task including all targets except `t_BottomCD` and `t_NeckingCD`. This can be done to fine-tune the targets without `t_BottomCD` and `t_NeckingCD` that did not match the expected accuracy after the execution of task `PMC_HAR`.

In the early stages of the calibration, it might be interesting to explore the calibration parameter space using the `explore` task and to analyze the inferred machine-learning data:

```
pmccal.explore_ml(task_name="PMC_HAR_1", tolerance=0.07,
                  strategy=0, num_parameters=0, constraint="Fail==0",
                  parent="../parent/HARC", child_dir="../ML1")

pmccal.infer_ml_models("PMC_HAR", child_dir="../ML1", doe_type="uniform",
                       doe_size=30000)
```

The task `PMC_HAR_1` performs an `explore` task. In this example, you might not be interested in the calibrated parameters from the `PMC_HAR_1` task but prefer to generate data using model inference (see Saving and Analyzing Machine-Learning Model Inference Data on page 77) and to analyze this data using a tool such as Sentaurus PCM Studio.

Validating Parameter Ranges

Parameter ranges are critical for the creation of a machine-learning model. If the parameter exploration space is too big, then the combination of parameters could lead to failures in simulations and, subsequently, to insufficient data for model training. The `validate_range_ml()` command executes the validation of the machine learning–based parameter range (see Validating Parameter Ranges on page 23).

The new ranges can be used in a subsequent task, since the parameter ranges are reset automatically, as shown in the following example:

```
pmccal.validate_range_ml(task_name="PMC_HAR", constraint="Fail==0",
                         parent="../parent/HARC", child_dir="../ML1")

pmccal.calibrate_har_etch_ml(task_name="PMC_HAR_1", tolerance=0.07,
                             num_parameters=0, constraint="Fail==0",
                             parent="../parent/HARC", child_dir="../ML1")
```

The task `PMC_HAR_1` uses the parameter ranges computed in task `PMC_HAR`.

Saving and Analyzing Machine-Learning Model Inference Data

The following command executes the model inference (see infer_ml_models on page 111):

```
pmccal.infer_ml_models(child_dir=<s>,
                       task_name=<s>,
                       doe=<s|d|p>,
                       doe_size=<i>,
                       doe_type=<s>,
                       doe_levels=<i>,
                       file_prefix=<s>,
                       doe_sep=<s>)
```

All exploration type of models, created during the execution of `task_name`, can be used to infer data. The design-of-experiments (DoE) used to generate the data can have the following properties:

- `doe` defines the range of features. If not specified, then the range is taken from the minimum and maximum values of the model features used during model training and creation. These values can be:
  - A dictionary specifying the minimum and maximum values for all or some features
  - A `.csv` file containing feature names as columns and corresponding values in the rows (in this case, minimum and maximum values are extracted from the values in the `.csv` file)
  - A pandas data frame, specifying the DoE range or the actual DoE if `doe_type="custom"`
- `doe_size` sets the number of rows of the DoE.
- `doe_type` sets the type of the DoE.
- `doe_levels` applies only to a full-factorial DoE (`doe_type="factorial"`). It defines the levels used for each model feature (two levels mean `min` and `max`).

The file created by `infer_ml_models()` is saved in the `child_dir` directory and has the following name format:

```
<file_prefix>_<task_name>_infer_<doe_type>_<index>.csv
```

The `index` depends on the number of available models created during the execution of `task_name`. Inference is performed on each model and the corresponding file is saved using a progressive index. The `.csv` file can be analyzed by a third-party application or Sentaurus PCM Studio.

Example

```
pmccal.calibrate_har_etch_ml(task_name="PMC_HAR", tolerance=0.07,
   num_parameters=0, constraint="Fail==0", parent="../parent/HARC",
   child_dir="../ML1")

pmccal.infer_ml_models(task_name="PMC_HAR", child_dir="../ML1",
   doe_type="nominal")

pmccal.infer_ml_models(task_name="PMC_HAR", child_dir="../ML1",
   doe_type="uniform", doe_size=30000)

pmccal.infer_ml_models(task_name="PMC_HAR", child_dir="../ML1",
   doe_type="uniform", doe_size=30000,
   doe="ML_Calibrate_range_1.csv",
   file_prefix="test_")
```

CHAPTER 7 Graphical User Interface

This chapter describes how to use the Sentaurus Calibration Workbench graphical user interface to create calibration flows.

The Sentaurus Calibration Workbench graphical user interface (GUI) is a specialized tool designed to provide an interface to the batch Sentaurus Calibration Workbench tool and to efficiently implement calibration strategies without involving the Python environment. It also can visualize the intermediate and final results.

Running the Graphical User Interface

You can run the Sentaurus Calibration Workbench GUI from the command line by using the command:

```
scwgui <options> <project_file>
```

where `<project_file>` is the name of a Sentaurus Calibration Workbench GUI project. All Sentaurus Calibration Workbench GUI project files are defined by the file extension `.scw`.

Table 4 Command-line options

Option	Description
`-h[elp]`	Displays help message
`-v[ersion]`	Displays version information

Features of the User Interface

The GUI has different panels (see Figure 9):

- The Projects panel, left side of the main window, shows all of the projects created or loaded, including the control and status of their execution (see Projects Panel).
- The Tables panel, in the middle of the main window, has different tabs (see Tables Panel).
- The Plots panel, right side of the main window, lists all of the plots you have generated, separated into tabs, one tab per project (see Plots Panel).

- The Output panel, at the bottom of the main window, shows the output of the execution of the Sentaurus Calibration Workbench batch tool that is executed by the Sentaurus Calibration Workbench GUI (see Output Panel).

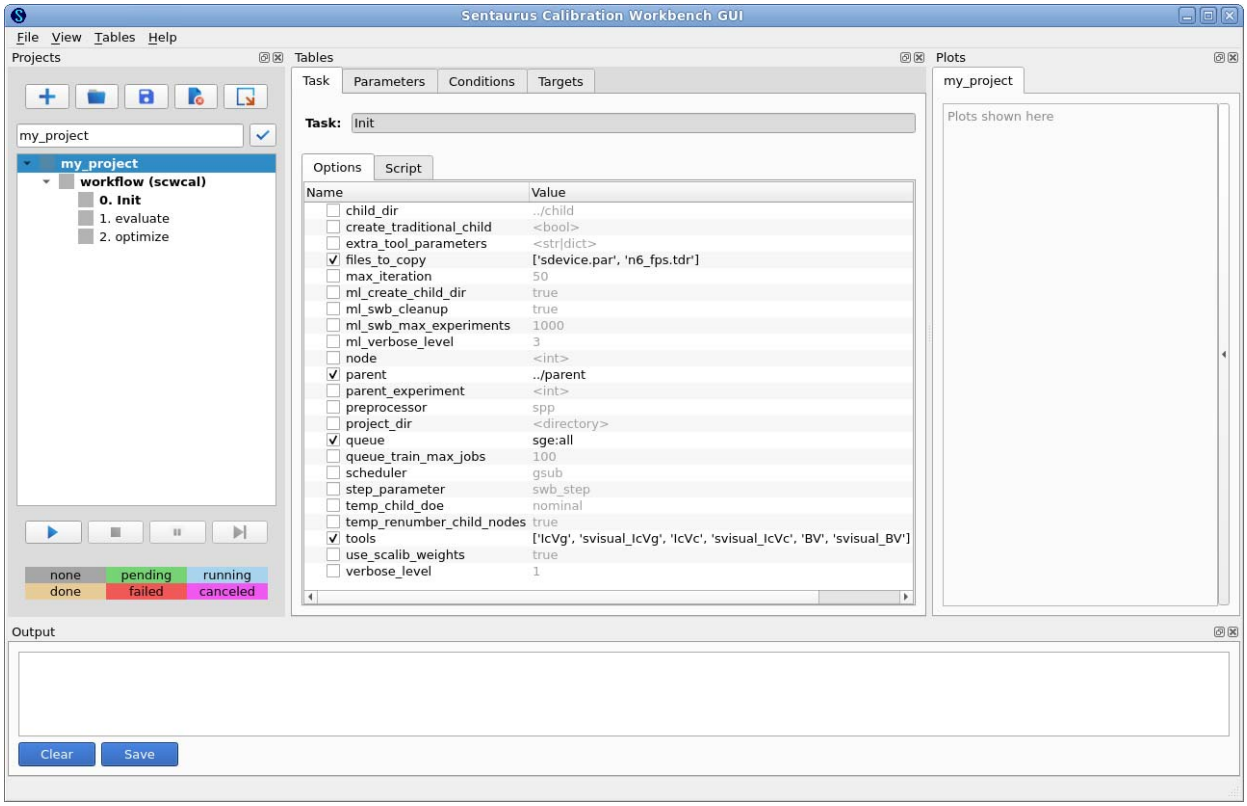

Figure 9 Main window of graphical user interface

Menu Bar

You can use the menu bar to access the main operations.

Table 5 Available menus

Menu	Description
File	Creates, loads, saves, closes, and exports Sentaurus Calibration Workbench GUI projects, and imports Sentaurus Workbench parameters.
View	Shows and hides panels; resets the layout.
Tables	Adds and removes rows from tables.
Help	Provides information about the Sentaurus Calibration Workbench GUI.

Projects Panel

This panel shows information about the different projects loaded and provides buttons to manage projects and their execution. This panel is divided into different parts:

- The File toolbar buttons allow you to interact with project files, for example, to create, open, save, close, and export files.
- The list of projects shows the strategy description of each calibration project, by dividing projects into workflows, and workflows into steps. Each of these elements shows the status of the calibration run by using different colors in the box next to each element.
- Under the list of projects, the Execution toolbar buttons allow you to control the execution of the calibration of each project. The status of the buttons depends on which project is selected.
- The color legend explains the status of a calibration run.

Bold text indicates the selected step and which step is represented in the Tables panel. If you select the project or the workflow, then the last executed step is bold (selected). When a project has not been executed, the Init step is selected.

Figure 10 Projects panel

Tables Panel

This panel presents information on the following tabs:

- The **Task** tab shows the *task definition* for a specific step (see Task Tab).
- The **Parameters** tab shows the *calibration parameters* and their attributes in tabular format (see Parameters Tab).
- The **Conditions** tab shows the *condition parameters* in tabular format (see Conditions Tab).
- The **Targets** tab shows the *targets* and their attributes in tabular format (see Targets Tab).

The information defined on these tabs is specific to each *calibration step*. If the text color of a value is gray, then it means that the value is inherited from a previous step. If a cell is orange, then it means that cell contains a calculated value as the result of the execution of the selected calibration step.

On the **Parameters**, **Conditions**, and **Targets** tabs, you can add or remove a row from the tables by clicking **Add** or **Remove** at the bottom of the tables, or by pressing Ctrl+M and Ctrl+Shift+M, respectively.

The tables have inline validation. Conflicting cells are pale red. The following are common conflicts:

- A cell has no value but a value is mandatory.
- A cell has a value that conflicts with another cell, for example, two cells colored pale red and both with values could mean that it is not valid for those two cells to have data.

Table cells might also be invalid for a specific condition, usually defined by the Type property. Such cells are colored dark gray.

Task Tab

This tab contains the details to define the task. It is defined by additional subtabs:

- The **Options** tab provides a set of defined options to be applied to a task. If none is defined, then the task uses the options from the previous step, or the default options if it is the Init step.
- The second subtab differs depending on the task used in the step: Init and custom tasks show the **Script** tab and, for other tasks, the **Arguments** tab is shown:
  - The **Script** tab allows you to define custom Python scripts.
  - The **Arguments** tab helps you to define the arguments for a given task, as well as setting its prologue and epilogue commands with their arguments.

Parameters Tab

The table contains a list of all *calibration parameters* used in the calibration project. The list must be defined in the Init step and cannot be changed in any of the consecutive steps. The table also contains the attributes of each calibration parameter, such as their lower and upper bounds.

Each element in the list (calibration parameter) can be included (checked) or excluded (unchecked) in each *calibration step*.

The Value column shows the current value of the parameter as a result of the execution of that step.

Conditions Tab

The table contains a list of all the *condition parameters* used in the calibration project with a defined value. The list must be defined in the Init step and cannot be changed in any of the consecutive steps, but their values can be changed.

Each element of the list (condition parameter) can be included (checked) or excluded (unchecked) in each *calibration step*.

Targets Tab

The table contains a list of all the targets used in the calibration project. The list must be defined in the Init step and cannot be changed in any of the consecutive steps. The table also contains the attributes of each target, such as their weight and scale.

Each element of the list (target) can be included (checked) or excluded (unchecked) in each *calibration step*.

The RMS Error and Value columns show the current RMS error and current simulated value (if a scalar target value has been defined) of the target as a result of the execution of that step.

Plots Panel

This panel contains all of the plots that you generated automatically or manually. The panel groups plots per project using tabs, where each tab is labeled by the project name.

On each tab, plots can have a grid layout or one maximized while others are hidden. Each tab can have an independent layout.

Each plot can be customized by modifying its properties, exported to a file, or closed, all independently of other plots.

Output Panel

This panel contains the output of the execution of the Sentaurus Calibration Workbench batch tool. Each project has its own Output panel, which changes according to the project selection in the Projects panel.

The output captured from the Sentaurus Calibration Workbench batch tool can be saved into a file or it can be deleted.

Setting Up a Calibration Flow

This section describes how to define a basic calibration flow using the Sentaurus Calibration Workbench GUI, following the setup explained in Chapter 2.

Setting up a calibration flow requires the following steps:

1. Create a new project and select the calibration module to use. See Creating a New Project and Selecting the Calibration Module.
2. Define the main calibration options on the **Options** tab of the **Task** tab for the Init step. See Setting Up the Calibration Input.
3. Add parameters, conditions, and targets to the Init step. See Setting Up the Calibration Input.
4. Add new steps, assigning a task to each one. Set up the task arguments and options. See Setting Up the Calibration Input.
5. Execute the calibration flow. See Executing the Calibration Flow.

Creating a New Project and Selecting the Calibration Module

You can select a calibration module when you create a Sentaurus Calibration Workbench GUI project (see Figure 11) and when additional workflows are added to an existing project.

This will create a project with the given name. The new project contains one workflow with one step, the Init step (see Figure 10).

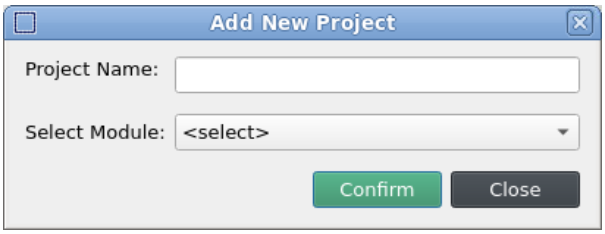

Figure 11 Add New Project dialog box where calibration module is selected

Setting Up the Calibration Input

Calibration input includes parameters, targets, the simulation flow (calibration steps), and its options.

In the Sentaurus Calibration Workbench GUI, parameters (calibration parameters), conditions (condition parameters), and targets are defined *only* in the Init step.

Options

You must define the *parent* task option in the Init step. You also can define several optional calibration controls. All of these options are located on the **Options** tab of the **Task** tab.

Some of these controls such as `parent`, `child_dir`, and `tools` are defined in Setting Options for Calibrations on page 13.

Parameters

Each parameter (calibration parameter) is a row in the table. You can add parameters on the **Parameters** tab by clicking **Add Parameter** (below the table). After a new row is added, you must define a name for the parameter and assign an initial value, which is mandatory.

The name of the parameter must correspond to a parameter name as defined in the Sentaurus Workbench project (case sensitive) and the parameter in the simulator command file or parameter file. You must use the Sentaurus Workbench parameter syntax (`@name@`) for parameters in parameterized files.

You can include or exclude parameters from a calibration task (step) by selecting or clearing the check box next to the parameter name.

Conditions

Each condition (condition parameter) is a row in the table. You can add conditions on the **Conditions** tab by clicking **Add Condition** (below the table). After a new row is added, you must define a name for the condition and assign a value.

The conditions are used in targets as split conditions.

Targets

Each target is a row in the table. You can add targets on the **Targets** tab by clicking **Add Target** (below the table). After a new row is added, you must define a name for the target and assign a target value, which can be a scalar for a scalar target or a reference target xy profile (one of them must be defined).

The name of the target must correspond to the name of a Sentaurus Workbench extracted variable (case sensitive).

You can include or exclude targets from a calibration task (step) by selecting or clearing the check box next to the target name.

If split conditions (condition parameters that can have different values) are defined, then you can use them by setting them in the Conditions column.

Tasks

Setting up tasks (steps) defines the calibration flow. You can add tasks by right-clicking the corresponding workflow (in the Projects panel) and choosing **Add New Step to Workflow**.

The step is added to the end of the workflow. Each module has a list of valid tasks that can be added as calibration steps (see Figure 12). If the step name is not defined, then it takes the same name as the task.

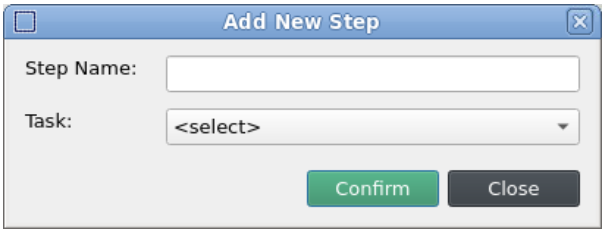

Figure 12 Add New Step dialog box

After a task is added, you can define its options. If the task is custom, then you can define a Python script to be executed in that task. Otherwise, for any other task type, you can define the

arguments of that task, as well as the prologue and epilogue commands (of that task) together with their arguments.

Executing the Calibration Flow

After you have defined the calibration flow, you execute the corresponding calibration by clicking the play button of the Execution toolbar in the Projects panel.

This procedure launches a new process and executes the Sentaurus Calibration Workbench batch tool (binary `scw`). The status of the Sentaurus Calibration Workbench GUI is updated at each step, including the last calculated values of parameters and targets.

To check the status of steps, refer the colored box next to the step name to the legend in the Projects panel.

In the parameters and targets tables, the last values calculated are shown in cells with an orange background (see Figure 13).

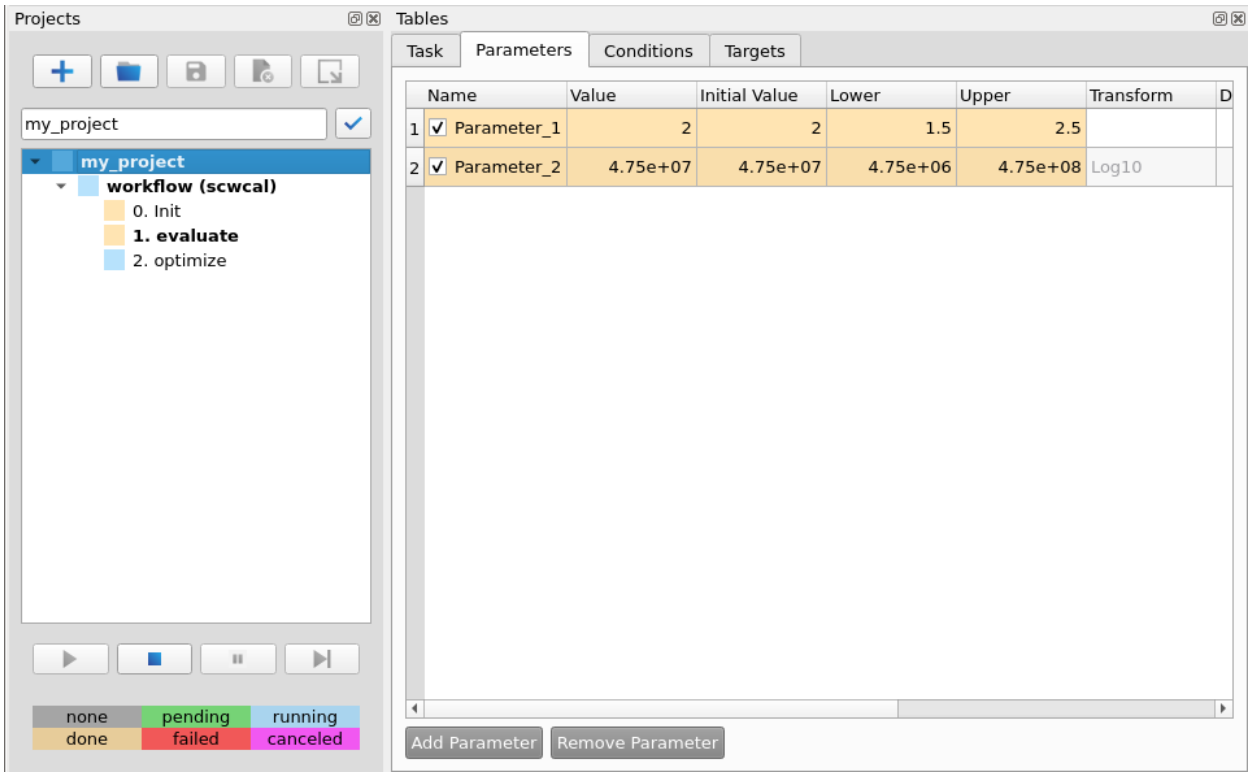

Figure 13 Projects panel and Tables panel showing a running project

APPENDIX A Commands

This appendix describes the commands of the Sentaurus Calibration Workbench calibration modules.

Syntax Conventions

The following conventions are used in the syntax of Python commands:

- `<a>`: Array of (x,y) type, for example, ((0.0, 1e19), (0.1, 1.3e19), (0.2, 1e18))
- `<b>`: Boolean, either `True` or `False`
- `<d>`: Dictionary
- `<f>`: Floating-point number
- `<i>`: Integer
- `<k>`: Keyword or type
- `<o>`: Object
- `<p>`: pandas data frame
- `<s>`: String value
- `<t>`: Tuple or list
- `()` denotes a grouping only
- `[]` denotes optional arguments
- | denotes exclusive options

NOTE Standard Python syntax conventions are followed. Command arguments can be written using either `<argument>=<value>` or `<value>`. For example:

```
cal.evaluate(parameters=[p1, p2, p3], targets=[t1, t2, t3],
   parent="../parent/example")

cal.evaluate([p1, p2, p3], [t1, t2, t3], "../parent/example")
```

If only values are used, then the arguments must be written in the order presented in the respective command syntax.

Only the `set_option()` command must be written using the format `<argument>=<value>`. This command does not accept value-only input.

calculate_r2

This command computes the *r*-squared of targets, which is the square of the Pearson correlation coefficient:

$r^2 = \left(\frac{C_{1,2}}{C_{1,1}C_{2,2}}\right)^2$ where C is the covariance matrix Cov[X,Y].

Syntax

```
calculate_r2(targets=<t>, [group_by_name=<b>])
```

Argument	Type	Description
`targets`	`<t>`	Specifies a list of targets.
`group_by_name`	`<b>`	Specifies how targets are grouped. If set to `True`, then the given targets are grouped by their names, and r^2 is calculated separately. Default: `False`

Returns

None

Examples

```
cal.calculate_r2(targets=[t1, t2, t3], group_by_name=True)
```

calibrate_ac

This command sets the AC characteristics calibration strategy, specific to the `igbtcal` calibration module (see IGBT AC Characteristics Calibration on page 66).

Syntax

```
calibrate_ac(parameters=<t>, targets=<t>,
             [parent=<s>,] [child_dir=<s>,]
             [tolerance_cv=<f>,] [tolerance_qg=<f>,]
             [tolerance_switching=<f>,] [max_iteration=<i>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`tolerance_cv`	`<f>`	Sets the RMS error tolerance for a `CV` calibration. Default: `0.0`
`tolerance_qg`	`<f>`	Sets the RMS error tolerance for a `Qg` calibration. Default: `0.0`
`tolerance_switching`	`<f>`	Sets the RMS error tolerance for a `Switching` calibration. Default: `0.0`
`max_iteration`	`<i>`	Sets the maximum number of final calibration loops. Default: `None` (uses the `max_iteration` set by `set_option(max_iteration=<i>)` or its default, 50)

Returns

None

Examples

```
igbtcal.calibrate_ac(parameters=parameters, targets=targets, max_iteration=30)
```

calibrate_dc

This command sets the DC characteristics calibration strategy, specific to the `igbtcal` calibration module (see IGBT DC Characteristics Calibration on page 65).

Syntax

```
calibrate_dc(parameters=<t>, targets=<t>,
             [parent=<s>,] [child_dir=<s>,]
             [tolerance_icvg=<f>,] [tolerance_icvc=<f>,]
             [tolerance_bv=<f>,] [max_iteration=<i>,]
             [doe_method=<s>,] [doe_nsamples=<i>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
tolerance_icvg	<f>	Sets the RMS error tolerance for an IcVg calibration. Default: 0.0
tolerance_icvc	<f>	Sets the RMS error tolerance for an IcVc calibration. Default: 0.0
tolerance_bv	<f>	Sets the RMS error tolerance for a BV calibration. Default: 0.1
max_iteration	<i>	Sets the maximum number of final calibration loops. Default: None (uses the max_iteration set by set_option(max_iteration=<i>) or its default, 50)
doe_method	<s>	Specifies the design-of-experiments (DoE) method for BV search before calibration. Default: "sobol"
doe_nsamples	<i>	Specifies the number of samples for DoE search before optimization. Default: None (uses the recommended nsample by the igbtcal module based on the number of parameters)

Returns

None

Examples

```
igbtcal.calibrate_dc(parameters=parameters, targets=targets, max_iteration=30,
                     doe_nsamples=20)
```

calibrate_har_etch

This command sets the calibration strategy, specific to the `pmccal` calibration module, based on a gradient-based local calibration algorithm for Sentaurus Topography 3D high aspect ratio etching (see HARC Etching Calibration on page 69).

Syntax

```
calibrate_har_etch(parameters=<t>, targets=<t>,
                   [parent=<s>,] [child_dir=<s>,]
                   [tolerance_end=<f>,] [tolerance_mid=<f>,]
                   [tolerance_init=<f>,] [max_iteration_end=<i>,]
                   [max_iteration_mid=<i>,] [max_iteration_init=<i>,]
                   [weight_factor=<f>,] [reevaluate=<b>,]
                   [use_bi_sensitivity=<b>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`tolerance_end`	`<f>`	Sets the average relative error tolerance for the final calibration. Default: `0.1`
`tolerance_mid`	`<f>`	Sets the average relative error tolerance for the middle calibration. Default: `0.14`
`tolerance_init`	`<f>`	Sets the average relative error tolerance for the initial calibration. Default: `0.05`
`max_iteration_end`	`<i>`	Sets the maximum number of final calibration loops. Default: `4`
`max_iteration_mid`	`<i>`	Sets the maximum number of middle calibration loops. Default: `2`
`max_iteration_init`	`<i>`	Sets the maximum number of initial calibration loops. Default: `5`
`weight_factor`	`<f>`	Specifies the factor by which the predefined weights of the `BottomZ` and `RemainingHM` targets are multiplied. Default: `2.0`
`reevaluate`	`<b>`	Specifies whether to reevaluate by rerunning the simulation due to variability.
`use_bi_sensitivity`	`<b>`	Specifies whether to switch on the bidirectional sensitivity calculation for the initial calibration. Default: `True`

Returns

None

Examples

```
pmccal.calibrate_har_etch(parameters=parameters, targets=targets,
                          tolerance_end=0.15)
```

calibrate_har_etch_ml

This command sets the calibration strategy, specific to the `pmccal` calibration module, based on machine learning for Sentaurus Topography 3D high aspect ratio etching (see HARC Etching Calibration on page 69).

Syntax

```
calibrate_har_etch_ml(parameters=<t>, targets=<t>,
                      [parent=<s>,] [child_dir=<s>,]
                      [task_name=<s>,] [tolerance=<f>,]
                      [num_parameters=<i>,] [num_explore_parameters=<i>,]
                      [num_refine_parameters=<i>,] [num_iterations=<i>,]
                      [explore_doe_range=<f|i>,] [explore_doe_method=<s>,]
                      [explore_doe_size=<i>,] [num_retries=<i>,]
                      [constraint=<s>,] [weight_factor=<f>,]
                      [validate_range=<b>,] [sensitivity=<b|s|d>,]
                      [tolerance_refine=<f>,] [priority_targets=<t>,]
                      [strategy=<s>,] [use_checkpoint=<b>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`task_name`	`<s>`	Sets the name of the machine-learning task. Default: `"PMC_HAR"`
`tolerance`	`<f>`	Sets the total relative RMS error tolerance. Default: `0.12`
`num_parameters`	`<i>`	Sets the number of primary parameters used for machine learning. If `num_parameters=0`, then all defined parameters are assumed to be primary parameters. Default: `0`

Argument	Type	Description
num_explore_parameters	<i>	Sets the number of primary parameters used for exploration. If num_explore_parameters=0, then all defined parameters are assumed to be primary parameters. Default: None
num_refine_parameters	<i>	Sets the number of primary parameters used for refinement. If num_refine_parameters=0, then all defined parameters are assumed to be primary parameters. Default: None
num_iterations	<i>	Sets the maximum number of iterations. Default: 1000
explore_doe_range	<f|s>	Sets the parameter range scaling used for the DoE. If a file name is provided, then it sets the name of a CSV file that specifies the parameter ranges for the DoE. Default: None
explore_doe_method	<s>	Sets the DoE method. Options are "box_behnken", "full_factorial", "latin_hypercube", "random", and "sobol".
explore_doe_size	<i>	Sets the number of samples for the DoE in the case of the DoE methods "latin_hypercube", "random", and "sobol". Default: None
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""
weight_factor	<f>	Sets the weight factor for the BottomZ and RemainingHMZ targets. Default: 2
validate_range	<b>	Specifies whether to perform a parameter range validation analysis. Default: True
sensitivity	<b|s|d>	Specifies the type of sensitivity analysis to be performed: • If True, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: None
tolerance_refine	<f>	Sets the tolerance value for the machine-learning refinement algorithm. Default: tolerance + 0.03
priority_targets	<t>	Specifies a list of priority targets in order of priority, starting with the highest priority. Default: None
strategy	<s>	Specifies the strategy. For good initial parameter values and ranges, use "standard". If the initial parameters are far from optimal or the nominal simulation failed, then use "validate". Default: "validate"

Argument	Type	Description
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False

Returns

None

Examples

```
pmccal.calibrate_har_etch_ml(parameters=parameters, targets=targets,
                             tolerance=0.1, explore_doe_range=0.5,
                             num_parameters=10)
```

calibrate_hcsd_nmos

This command sets the calibration command for NMOS devices, specific to the `devcal` calibration module (see Hot-Carrier Stress Degradation Calibration on page 57).

Syntax

```
calibrate_hcsd_nmos(parameters=<t>, targets<t>,
                    [use_thermal=<b>,] [condition_variable=<s>,]
                    [parent=<s>,] [child_dir=<s>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of two calibration parameter objects, in the following order: 1. Sentaurus Device acceptor trap distribution central energy 2. Sentaurus Device acceptor trap distribution energy width
`targets`	`<t>`	Specifies a list of calibration target objects.
`use_thermal`	`<b>`	Specifies whether to use the field-enhanced thermal component of the HCS degradation model in Sentaurus Device. Default: `True`
`condition_variable`	`<s>`	Sets the condition variable of the targets, in particular, the transient final time. Default: `"time"`
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.

Returns

None

Examples

```
devcal.calibrate_hcsd_nmos(parameters=[EmidA, EsigA],
                           targets=[t_1_IdVg, t_2_IdVg, t_3_IdVg, t_e_IdVg],
                           condition_variable="Time")
```

calibrate_hcsd_pmos

This command sets the calibration command for PMOS devices, specific to the `devcal` calibration module (see Hot-Carrier Stress Degradation Calibration on page 57).

Syntax

```
calibrate_hcsd_pmos(parameters=<t>, targets<t>,
                    [condition_variable=<s>,]
                    [parent=<s>,] [child_dir=<s>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of four calibration parameter objects, in the following order: 1. Sentaurus Device acceptor trap distribution central energy 2. Sentaurus Device acceptor trap distribution energy width 3. Sentaurus Device donor trap distribution central energy 4. Sentaurus Device donor trap distribution energy width
`targets`	`<t>`	Specifies a list of calibration target objects.
`condition_variable`	`<s>`	Sets the condition variable of the targets, in particular, the transient final time. Default: `"time"`
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.

Returns

None

Examples

```
devcal.calibrate_hcsd_pmos(parameters=[EmidA, EsigA, EmidD, EsigD],
                           targets=[t_1_IdVg, t_2_IdVg, t_3_IdVg, t_e_IdVg],
                           condition_variable="Time")
```

calibrate_ml

This command sets the calibration strategy based on machine learning.

```
calibrate_ml(parameters=<t>, targets=<t>,
             [parent=<s>,] [child_dir=<s>,]
             [task_name=<s>,] [tolerance=<f>,]
             [num_parameters=<i>,] [num_explore_parameters=<i>,]
             [num_refine_parameters=<i>,] [num_iterations=<i>,]
             [explore_doe_range=<f>,] [explore_doe_method=<s>,]
             [explore_doe_size=<i>,] [num_retries=<i>,]
             [constraint=<s>,] [validate_range=<b>,]
             [sensitivity=<b|s|d>,] [tolerance_refine=<f>,]
             [priority_targets=<t>,] [strategy=<s>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
task_name	<s>	Sets the name of the machine-learning task. Default: "ML_calibrate"
tolerance	<f>	Sets the total relative RMS error tolerance. Default: 0.12
num_parameters	<i>	Sets the number of primary parameters used for machine learning. If num_parameters=0, then all defined parameters are assumed to be primary parameters. Default: 0
num_explore_parameters	<i>	Sets the number of primary parameters used for exploration. If num_explore_parameters=0, then all defined parameters are assumed to be primary parameters. Default: None
num_refine_parameters	<i>	Sets the number of primary parameters used for refinement. If num_refine_parameters=0, then all defined parameters are assumed to be primary parameters. Default: None
num_iterations	<i>	Sets the maximum number of iterations. Default: 1000
explore_doe_range	<f>	Sets the parameter range scaling used for the DoE. If a file name is provided, then it sets the name of a CSV file that specifies the parameter ranges for the DoE. Default: None
explore_doe_method	<s>	Sets the DoE method. Options are "box_behnken", "full_factorial", "latin_hypercube", "random", and "sobol".
explore_doe_size	<i>	Sets the number of samples for the DoE in the case of the DoE methods "latin_hypercube", "random", and "sobol". Default: None

Argument	Type	Description
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""
validate_range	<b>	Specifies whether to perform a parameter range validation analysis. Default: True
sensitivity	<b|s|d>	Specifies the type of sensitivity analysis to be performed: • If True, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: None
tolerance_refine	<f>	Sets the tolerance value for the machine-learning refinement algorithm. Default: tolerance + 0.03
priority_targets	<t>	Specifies a list of priority targets in order of priority, starting with the highest priority. Default: None
strategy	<s>	Specifies the strategy. For good initial parameter values and ranges, specify "standard". If the initial parameters are far from optimal or the nominal simulation failed, then specify "validate". Default: "validate"

Returns

None

Examples

```
cal.calibrate_ml(parameters=parameters, targets=targets, tolerance=0.1,
                 num_parameters=10)
```

determine_primary_parameters

This command determines the primary parameters based on sensitivity analysis, following a compute sensitivity task `run_sensitivity_analysis()` (see run_sensitivity_analysis on page 121). Primary parameters are defined as parameters with high sensitivity. Secondary parameters are defined as parameters with low sensitivity.

Syntax

```
determine_primary_parameters(parameters=<t>,
                             [num_parameters=<i>,]
                             [min_sensitivity=<f>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects for which a sensitivity analysis has been performed previously.
`num_parameters`	`<i>`	Sets the maximum number of primary parameters. The minimum number is 1. Default: `6`
`min_sensitivity`	`<f>`	Sets the sensitivity of only primary parameters, which must be greater than `min_sensitivity`. Default: `0.001`

Returns

`<t>`: List of primary parameter objects only

Examples

```
cal.determine_primary_parameters(parameters=parameters, num_parameters=5)
```

determine_primary_parameters_ml

This command determines the primary parameters based on sensitivity analysis, following a compute sensitivity task `run_sensitivity_analysis_ml()`. Primary parameters are defined as parameters with high sensitivity. Secondary parameters are defined as parameters with low sensitivity.

Syntax

```
determine_primary_parameters_ml(parameters=<t>,
                                [task_name=<s>,]
                                [child_dir=<s>,]
                                [num_parameters=<i>,]
                                [min_sensitivity=<f>,]
                                [targets=<t>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects for which a sensitivity analysis has been performed previously.
`task_name`	`<s>`	Sets the name of the task that is used in the earlier performed `run_sensitivity_analysis_ml(task_name=<s>)`. Default: `"ML_sensitivity"`
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`num_parameters`	`<i>`	Sets the maximum number of primary parameters. The minimum number is 1. Default: `8`
`min_sensitivity`	`<f>`	Sets the sensitivity of only primary parameters, which must be greater than `min_sensitivity`. Default: `0.001`
`targets`	`<t>`	Specifies a list of targets to be included. Default: `None`

Returns

`<t>`: List of primary parameter objects only

Examples

```
cal.determine_primary_parameters_ml(parameters=parameters, num_parameters=5)
```

evaluate

This command performs a simple evaluation to calculate errors, without calibration. See Simple Evaluation on page 16.

Syntax

```
evaluate(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>,]
         [extra_tool_parameters=<s|d>])
```

Syntax for pmccal Module

```
evaluate(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>,]
         [extra_tool_parameters=<s|d>,] [reevaluate=<b>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
extra_tool_parameters	<s|d>	Specifies extra tool parameters to be added to the child project. If set to None, then option("extra_tool_parameters") is applied. If still set to None (default), then no parameters are added. If it specifies a string, then all parameters are added to this tool. If it specifies a dictionary, then parameters are added to individual tools. Default: None
reevaluate	<b>	(For pmccal module only) Specifies whether to reevaluate by rerunning the simulation due to variability. Default: True

Returns

None

Examples

```
cal.evaluate(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

```
cal.evaluate(parameters=[p1, p2, p3], targets=[t1, t2, t3]
   parent="../parent/example")
```

evaluate_ac

This command sets the AC characteristics evaluation to calculate errors, without calibration, specific to the `igbtcal` calibration module (see IGBT AC Characteristics Calibration on page 66.)

Syntax

```
evaluate_ac(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.

Returns

None

Examples

```
igbtcal.evaluate_ac(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

```
igbtcal.evaluate_ac(parameters=[p1, p2, p3], targets=[t1, t2, t3]
                    parent="../parent/example")
```

evaluate_dc

This command sets the DC characteristics evaluation to calculate errors, without calibration, specific to the `igbtcal` calibration module (see IGBT DC Characteristics Calibration on page 65).

Syntax

```
evaluate_dc(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.

Returns

None

Examples

```
igbtcal.evaluate_dc(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

```
igbtcal.evaluate_dc(parameters=[p1, p2, p3], targets=[t1, t2, t3]
                    parent="../parent/example")
```

explore_har_etch_ml

This command explores a machine-learning model, specific to the `pmccal` calibration module, for Sentaurus Topography 3D high aspect ratio etching (see HARC Etching Calibration on page 69).

Syntax

```
explore_har_etch_ml(parameters=<t>, targets=<t>,
                    [parent=<s>,] [child_dir=<s>,]
                    [task_name=<s>,] [tolerance=<f>,] [num_parameters=<i>,]
                    [explore_doe_range=<f|s>,] [explore_doe_method=<s>,]
                    [explore_doe_size=<i>,] [num_retries=<i>,]
                    [constraint=<s>,] [weight_factor=<f>,]
                    [sensitivity=<b|s|d>,] [priority_targets=<t>,]
                    [use_checkpoint=<b>,] [num_iterations=<i>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`task_name`	`<s>`	Sets the name of the machine-learning task. Default: `"ML_explore"`
`tolerance`	`<f>`	Sets the total relative RMS error tolerance. Default: `0.12`
`num_parameters`	`<i>`	Sets the number of primary parameters used for machine learning. If `num_parameters=0`, then all defined parameters are assumed to be primary parameters. Default: `0`
`explore_doe_range`	`<f|s>`	Sets the parameter range scaling used for the DoE. If a file name is provided, then it sets the name of a CSV file that specifies the parameter ranges for the DoE. Default: `None`
`explore_doe_method`	`<s>`	Sets the DoE method. Options are `"box_behnken"`, `"full_factorial"`, `"latin_hypercube"`, `"random"`, and `"sobol"`.
`explore_doe_size`	`<i>`	Sets the number of samples for the DoE in the case of the DoE methods `"latin_hypercube"`, `"random"`, and `"sobol"`. Default: `None`
`num_retries`	`<i>`	Sets the number of simulation run retries if there is a simulation runtime failure. Default: `1`
`constraint`	`<s>`	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: `""`

Argument	Type	Description
weight_factor	<f>	Sets the weight factor for the BottomZ and RemainingHMZ targets. Default: 2
sensitivity	<b|s|d>	Specifies the type of sensitivity analysis to be performed: • If True, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: None
priority_targets	<t>	Specifies a list of priority targets in order of priority, starting with the highest priority. Default: None
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False
num_iterations	<i>	Specifies how many exploration steps are performed. Default: 1

Returns

None

Examples

```
pmccal.explore_har_etch_ml(parameters=parameters, targets=targets,
                           tolerance=0.1, explore_doe_range=0.5,
                           num_parameters=10)
```

explore_ml

This command explores a machine-learning model.

Syntax

```
explore_ml(parameters=<t>, targets=<t>,
           [parent=<s>,] [child_dir=<s>,]
           [task_name=<s>,] [tolerance=<f>,] [num_parameters=<i>,]
           [explore_doe_range=<f|s>,] [explore_doe_method=<s>,]
           [explore_doe_size=<i>,] [num_retries=<i>,] [constraint=<s>,]
           [sensitivity=<b|s|d>,] [priority_targets=<t>,]
           [use_checkpoint=<b>,] [num_iterations=<i>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
task_name	<s>	Sets the name of the machine-learning task. Default: "ML_explore"
tolerance	<f>	Sets the total relative RMS error tolerance. Default: 0.12
num_parameters	<i>	Sets the number of primary parameters used for machine learning. If num_parameters=0, then all defined parameters are assumed to be primary parameters. Default: 0
explore_doe_range	<f|s>	Sets the parameter range scaling used for the DoE. If a file name is provided, then it sets the name of a CSV file that specifies the parameter ranges for the DoE. Default: None
explore_doe_method	<s>	Sets the DoE method. Options are "box_behnken", "full_factorial", "latin_hypercube", "random", and "sobol". Default: None
explore_doe_size	<i>	Sets the number of samples for the DoE in the case of the DoE methods "latin_hypercube", "random", and "sobol". Default: None
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""

Argument	Type	Description
sensitivity	<b|s|d>	Specifies the type of sensitivity analysis to be performed: • If True, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: None
priority_targets	<t>	Specifies a list of priority targets in order of priority, starting with the highest priority. Default: None
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False
num_iterations	<i>	Specifies how many exploration steps are performed. Default: 1

Returns

None

Examples

```
cal.explore_ml(parameters=parameters, targets=targets, tolerance=0.1,
               num_parameters=10, num_iterations=5)
```

get_rms_error

This command returns the current RMS error for a specific target or for a weighted total.

Syntax

```
get_rms_error([target=<s>,] [iteration=<i>])
```

Argument	Type	Description
target	<s>	Specifies the name of a target. If set to None, then the total weighted RMS error is returned. Default: None
iteration	<i>	Specifies the iteration number for which the RMS error is returned. The default, iteration=0, means the optimum iteration with minimum RMS error.

Returns

<f>: RMS error

Examples

```
cal.get_rms_error()
```

```
cal.get_rms_error(target="t_BottomZ")
```

get_step_signature

This command returns the current calibration step or task signature.

Syntax

```
get_step_signature()
```

Returns

<i>: Calibration step signature or step number

infer_ml_models

This command sets up the machine-learning model inference.

Syntax

```
infer_ml_models(child_dir=<s>, [task_name=<s>,]
                [doe=<s|d|p>,] [doe_size=<i>,] [doe_type=<s>,]
                [doe_levels=<i>,] [file_prefix=<s>,] [doe_sep=<s>])
```

Argument	Type	Description
child_dir	<s>	Sets the directory of the child projects.
task_name	<s>	Specifies the name of a task from which the models used for inference are loaded. Default: "ML_Calibrate"
doe	<s|d|p>	Sets the parameters range that overrides the model defaults. The value can be a string specifying the file name to use, a dictionary, or a pandas data frame. Default: None
doe_size	<i>	Sets the number of rows in the generated DoE for the inference. Default: 10000
doe_type	<s>	Sets the type of DoE. Options are: • "uniform": Random uniform distribution • "box_behnken" • "centered_uniform" • "custom" • "full_factorial": Full-factorial • "nominal" • "OFAT": One-factor-at-a-time Default: "uniform"
doe_levels	<i>	Sets the parameter levels used in a full-factorial DoE (doe_type="factorial"). It defines the levels used for each model feature (two levels mean min and max). Default: 2
file_prefix	<s>	Sets the prefix to be used in the created file names. Default: ""
doe_sep	<s>	Specifies the separator to be used when creating files. Default: None

Returns

None

Examples

```
cal.infer_ml_models(child_dir="../path/to/child/directory",
                    task_name="ML_explore", doe_type="nominal")
```

optimize

This command performs the gradient-based local calibration task.

Syntax

```
optimize(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>,]
         [max_iteration=<i>])
```

Syntax for pmccal Module

```
optimize(parameters=<t>, targets=<t>, [parent=<s>,] [child_dir=<s>,]
         [max_iteration=<i>,] [reevaluate=<b>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`max_iteration`	`<i>`	Sets a limit to the maximum number of iterations for the calibration. Default: `None`, which specifies to set to `max_iteration` in `set_option()`
`reevaluate`	`<b>`	(For `pmccal` module only) Specifies whether to reevaluate by rerunning the simulation due to variability. Default: `True`

Returns

None

Examples

```
cal.optimize(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

```
cal.optimize(parameters=parameters, targets=targets, max_iteration=25)
```

Parameter

This class represents a model parameter to be calibrated.

General Syntax

```
Parameter(name=<s>, value=<f>, minimum=<f>, maximum=<f>,
          [transform=<o>,] [choices=<t>,] [sort=<b>,] [type=<k>,] [dtype=<k>,]
          [precision_rel=<i>,] [precision_abs=<i>,] [force_pass=<b>])
```

Argument	Type	Description
name	<s>	Specifies the name of a calibration parameter. This must be the name of a Sentaurus Workbench parameter to be calibrated.
value	<f>	Specifies the initial value or the default of the parameter.
minimum	<f>	Specifies the minimum-allowed value for the parameter.
maximum	<f>	Specifies the maximum-allowed value for the parameter.
transform	<o>	Specifies the transform object for the parameter value. The transformed value is "transform(value)".
choices	<t>	Specifies the candidate values for a discrete parameter.
sort	<b>	Specifies whether to sort the discrete parameter values before encoding. Default: False
type	<k>	Specifies an optional type for parameter classification. Available types for the igbtcal module are IcVg, IcVg2, IcVc, IcVc2, BV, CV, Qg, and Switching.
dtype	<k>	Specifies the Python datatype of the parameter. Options are bool, float, int, or str. Default: float
precision_rel	<i>	Specifies the relative precision given as a number of significant digits. Default: 8
precision_abs	<i>	Specifies the absolute precision given as a number of digits after the decimal point (or before the decimal point if it is negative).
force_pass	<b>	(Discrete parameters only) Specifies whether samples are forced to pass all the values given by choices. Default: False

Returns

<o>: Parameter object

Examples

```
Parameter(name="p1", value=1.0, minimum=-10, maximum=10)

Parameter(name="p2", value=1e18, minimum=1e15, maximum=1e21, transform=Log10)
```

```
Parameter(name="p1", value=2.0, minimum=1.5, maximum=2.5, type=IcVg)

Parameter(name="p2", value=1.0, minimum=1E-4, maximum=1E4, transform=Log10,
          type=IcVc)
```

plot_error_history

This command generates a plot image with RMS error history after gradient-based local calibration.

Syntax

```
plot_error_history([file=<s>,] [targets=<t>,] [verbose=<b>])
```

Argument	Type	Description
file	<s>	Sets the name of an output file to be saved. Default: "error.png"
targets	<t>	Specifies a list of target objects for the error history plot. If set to None, then only the total error history is plotted. Default: None
verbose	<b>	Sets the verbosity level. Default: False

Returns

None

Examples

```
cal.plot_error_history(file="n@node@_error.png")
```

print_parameters

This command logs the calibrated parameters for Sentaurus Workbench variable extraction.

Syntax

```
print_parameters(parameters=<t>, [suffix=<s>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of parameter objects.
suffix	<s>	Specifies a suffix to append to the parameter name for the Sentaurus Workbench variable extraction. Default: '_cal'

Returns

None

Examples

```
cal.print_parameters(parameters=parameters)
```

print_targets

This command logs the calibrated targets for Sentaurus Workbench variable extraction.

Syntax

```
print_targets(targets=<t>, [suffix=<s>,] [include_path=<b>])
```

Argument	Type	Description
`targets`	`<t>`	Specifies a list of target objects.
`suffix`	`<s>`	Specifies a suffix to append to the target name for the Sentaurus Workbench variable extraction. Default: `'_cal'`
`include_path`	`<b>`	Specifies whether to include the path for targets linked to files. If set to `True`, then the path is included for targets linked to files.

Returns

None

Examples

```
cal.print_targets(targets=targets, include_path=True)
```

refine_har_etch_ml

This command refines a machine-learning model, specific to the `pmccal` calibration module, for Sentaurus Topography 3D etching (see Machine-Learning Model Refinement on page 74).

Syntax

```
refine_har_etch_ml(parameters=<t>, targets=<t>,
                   [parent=<s>,] [child_dir=<s>,]
                   [task_name=<s>,] [tolerance=<f>,] [num_parameters=<i>,]
                   [num_retries=<i>,] [constraint=<s>,]
                   [weight_factor=<f>,] [sensitivity=<b|s|d>,]
                   [use_checkpoint=<b>,] [num_iterations=<i>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`task_name`	`<s>`	Sets the name of the machine-learning task. Default: `"ML_refine"`
`tolerance`	`<f>`	Sets the total relative RMS error tolerance. Default: `0.12`
`num_parameters`	`<i>`	Sets the number of primary parameters used for machine learning. If `num_parameters=0`, then all defined parameters are assumed to be primary parameters. Default: `0`
`num_retries`	`<i>`	Sets the number of simulation run retries if there is a simulation runtime failure. Default: `1`
`constraint`	`<s>`	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: `""`
`weight_factor`	`<f>`	Sets the weight factor for the `BottomZ` and `RemainingHMZ` targets. Default: `2`
`sensitivity`	`<b|s|d>`	Specifies the type of sensitivity analysis to be performed: • If `True`, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: `None`

Argument	Type	Description
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False
num_iterations	<i>	Specifies how many refinement steps are performed. Default: 1

Returns

None

Examples

```
pmccal.refine_har_etch_ml(parameters=parameters, targets=targets,
                          tolerance=0.1, num_parameters=10)
```

refine_ml

This command refines a machine-learning model.

Syntax

```
refine_ml(parameters=<t>, targets=<t>,
          [parent=<s>,] [child_dir=<s>,]
          [task_name=<s>,] [tolerance=<f>,] [num_parameters=<i>,]
          [num_retries=<i>,] [constraint=<s>,] [sensitivity=<b|s|d>,]
          [use_checkpoint=<b>,] [num_iterations=<i>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
task_name	<s>	Sets the name of the machine-learning task. Default: "ML_refine"
tolerance	<f>	Sets the total relative RMS error tolerance. Default: 0.12
num_parameters	<i>	Sets the number of primary parameters used for machine learning. If num_parameters=0, then all defined parameters are assumed to be primary parameters. Default: 0
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""
sensitivity	<b|s|d>	Specifies the type of sensitivity analysis to be performed: • If True, then a sensitivity analysis is performed. • If a task name is provided, then the argument specifies the existing task to be used for sensitivity. • If a pandas data frame is provided, then the argument specifies the sensitivity information to be used. Default: None
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False

Argument	Type	Description
num_iterations	<i>	Specifies how many refinement steps are performed. Default: 1

Returns

None

Examples

```
cal.refine_ml(parameters=parameters, targets=targets, tolerance=0.1,
              num_parameters=10, num_iterations=5)
```

run_sensitivity_analysis

This command computes parameter sensitivity.

Syntax

```
run_sensitivity_analysis(parameters=<t>, targets=<t>,
                         [parent=<s>,] [child_dir=<s>,]
                         [percentage_range=<f>,] [num_points=<i>,]
                         [direction=<s>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
percentage_range	<f>	Specifies the range of the domain to be used for the sensitivity analysis. Default: None, taking sensdx from set_backend_control()
num_points	<i>	Specifies the number of values per parameter to simulate within the given range. Default: 3
direction	<s>	Specifies the direction for the increment of each parameter. Options are "both", "forward", and "backward". Default: "both"

Returns

None

Examples

```
cal.run_sensitivity_analysis(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

```
cal.run_sensitivity_analysis(parameters=parameters, targets=targets,
                             percentage_range=0.2)
```

run_sensitivity_analysis_ml

This command computes parameter sensitivity.

Syntax

```
run_sensitivity_analysis_ml(parameters=<t>, targets=<t>,
                            [parent=<s>,] [child_dir=<s>,]
                            [task_name=<s>,] [num_retries=<i>,]
                            [constraint=<s>,] [sensitivity=<s>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
task_name	<s>	Sets the name of the task. Default: "ML_sensitivity"
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""
sensitivity	<s>	Specifies the existing task to be used for sensitivity. Default: None

Returns

None

Examples

```
cal.run_sensitivity_analysis_ml(parameters=[p1, p2, p3], targets=[t1, t2, t3])
```

save_sensitivity

This command saves the sensitivity of parameters to a CSV file.

Syntax

```
save_sensitivity([path=<s>])
```

Argument	Type	Description
path	<s>	Specifies a path where sensitivity results are saved. Default: `'./n@node@_sensitivity.csv'`

Returns

None

Examples

```
cal.save_sensitivity(path='./sensitivity.csv')
```

search

This command performs an evaluation of different designs-of-experiments (DoE) to explore the parameter space.

Syntax

```
search(parameters=<t>, targets=<t>, method=<s>,
       [options=<d>,] [parent=<s>,] [child_dir=<s>,]
       [include_points=<t>,] [include_nominal=<b>])
```

Argument	Type	Description
`parameters`	`<t>`	Specifies a list of calibration parameter objects.
`targets`	`<t>`	Specifies a list of calibration target objects.
`method`	`<s>`	Sets the method used to perform the search (see Design-of-Experiments Search on page 19). Options are: • `"bayesian_optimizer"` creates experiments adaptively, starting from an initial set of experiments defined by Latin hypercube sampling. In each iteration, the algorithm selects some parameter sets that have the greatest potential to minimize the response. • `"grid"` creates experiments with parameter values defined on a regular grid. • `"latin"` creates experiments with parameter values using Latin hypercube sampling. • `"random"` creates experiments with random parameter values. • `"sobol"` creates experiments with parameter values using a Sobol sequence. • `"turbo"` creates experiments adaptively and builds on top of Bayesian optimization by keeping separate models that can change the focus of their search as the algorithm progresses. This makes it more likely to converge to a better local minimum. • The name of a Sentaurus Workbench DoE creates experiments according to the given Sentaurus Workbench DoE. Some of these methods have additional options that can be specified using the `options` argument. For details about these options, see the *Sentaurus™ Workbench Optimization Framework User Guide*, Searching the Parameter Space.
`options`	`<d>`	Specifies a dictionary of specific options for the search method.
`parent`	`<s>`	Specifies the parent project. If set to `None`, then `option("parent")` is used.
`child_dir`	`<s>`	Specifies the directory of the child projects. If set to `None`, then `option("child_dir")` is used.
`include_points`	`<t>`	Specifies a list of dictionaries defining points to be included in the search. Default: `None`
`include_nominal`	`<b>`	Specifies whether to include the current nominal in the search. Default: `True`

Returns

None

Examples

```
cal.search(parameters=[p1, p2], targets=[t1, t2], method="grid",
           options={"Ns": (2, 2), midpoint=True})

cal.search(parameters=parameters, targets=targets, method="sobol",
           options={"Ns": 10})
```

set_backend_control

This command defines the control parameters for the calibration backend.

Syntax

```
set_backend_control([initdx=<f>,] [sensdx=<f>,] [mindx=<f>,] [maxdx=<f>,]
                    [ssq_error_tolerance=<f>,] [parameter_tolerance=<f>,]
                    [target_tolerance=<f>,] [jacobian_options=<d>,]
                    [method=<s>,] [options=<d>])
```

Argument	Type	Description
initdx	<f>	Sets the initial increment of the normalized value for the calibration by the scalib backend. Default: 4e-2
sensdx	<f>	Sets the increment of the normalized value for sensitivity analysis. Default: 0.1
mindx	<f>	Sets the minimum increment of the normalized value for the Jacobian calculation by the scalib backend. Default: 1e-7
maxdx	<f>	Sets the maximum increment of the normalized value for the Jacobian calculation by the scalib backend. Default: 1.0
ssq_error_tolerance	<f>	Sets the tolerance of the normalized update of the sum of squares (ssq) error by the scalib backend. Default: 1e-5
parameter_tolerance	<f>	Sets the tolerance of the normalized parameter update by the scalib backend. Default: 2e-3
target_tolerance	<f>	Sets the tolerance of the total target error by the scalib backend. Default: 0.0
jacobian_options	<d>	Specifies a dictionary of keyword arguments passed to the Jacobian evaluation for backends other than the scalib backend.
method	<s>	Sets the calibration method used by backends other than the scalib backend.
options	<d>	Specifies a dictionary of backend-specific options for backends other than the scalib backend.

Returns

None

Examples

```
cal.set_backend_control(initdx=0.005)

cal.set_backend_control(sensdx=0.1, maxdx=1.0)

cal.set_backend_control(ssq_error_tolerance=1e-4)

cal.set_backend_control(parameter_tolerance=0.01)
```

set_backend_method

This command defines the calibration backend. It is only needed for backends other than the `scalib` backend, which is the default.

Syntax

```
set_backend_method(name=<s>)
```

Argument	Type	Description
`name`	`<s>`	Specifies the name of the calibration backend to be used. Options are: • `"scalib"` (default) • `"least_squares"` • `"minimize"` • `"mkl_nls"` • `"shgo"`

Returns

None

Examples

```
cal.set_backend_method(name="minimize")

cal.set_backend_method("minimize")
```

set_option

This command sets user-specified option values. The options listed here are general ones available for all calibration modules.

NOTE The `set_option()` command must be written using the format `<argument>=<value>`. This command does not accept value-only input.

Syntax

```
set_option([child_dir=<s>,] [extra_tool_parameters=<s|d>,]
           [files_to_copy=<t>,] [max_iteration=<i>,]
           [tools=<t>,] [queue=<s>,] [scheduler=<s>,]
           [parent=<s>,] [parent_experiment=<i>,]
           [verbose_level=<i>,] [use_scalib_weights=<b>])
```

Argument	Type	Description
`child_dir`	`<s>`	Specifies the child project directory. Default: `"../child/"`
`extra_tool_parameters`	`<s|d>`	Specifies the extra tool parameters to be added to the child project. If set to `None` (default), then no parameters are added. If it specifies a string, then all parameters are added to this tool. If it specifies a dictionary, then parameters are added to individual tools. Default: `None`
`files_to_copy`	`<t>`	Specifies a list of files and folders to be copied from the parent project to the child projects. Default: `None`
`max_iteration`	`<i>`	Sets the maximum number of iterations of the calibration. Default: `50`
`tools`	`<t>`	Sets the Sentaurus Workbench tool steps to be included in the calibration.
`queue`	`<s>`	Sets the queue name for executing the child projects.
`scheduler`	`<s>`	Sets the scheduler command. Default: `"gsub"`
`parent`	`<s>`	Specifies the parent project. Default: `None`
`parent_experiment`	`<i>`	Specifies the number of the nominal experiment of the parent project. Default: `None`
`verbose_level`	`<i>`	Sets the verbosity level. Setting `verbose_level=1` prints out whether the parameters are redundant, as well as the change to the sensitivity, dependence, and variability during calibration. Default: `0`
`use_scalib_weights`	`<b>`	Specifies whether to use the total weighted RMS error based on the `scalib` target weights. See Calculating the Total Weighted Root Mean Square Error on page 15. Default: `True`

Returns

None

Examples

```
cal.set_option(max_iteration=50)
```

set_step_signature

This command changes the calibration step signature.

Syntax

```
set_step_signature(signature=<s>)
```

Argument	Type	Description
signature	<s>	Sets the calibration step signature.

Returns

None

Examples

```
cal.set_step_signature(signature="1.0")
```

Target

This class represents a model target against which it is calibrated. The target can be a scalar or an xy profile.

Syntax for Scalar Targets

```
Target(name=<s>, value=<f>,
       [id=<s>,] [type=<k>,] [dtype=<k>,] [conditions=<d>,]
       [weight=<f>,] [scale=<f>,] [range=<t>,] [abs_value_range=<t>,]
       [values=<t>,] [dimension=<i>,] [error_method=<s>])
```

Argument	Type	Description
`name`	`<s>`	Specifies the name of a Sentaurus Workbench parameter or variable in which the simulation response will be available (as a scalar value).
`value`	`<f>`	Specifies the value to be matched by the simulation response. It must be a single scalar value. Default: `0.0`
`id`	`<s>`	Specifies a unique name or an ID of the target.
`type`	`<k>`	Specifies the type for target classification. Available types for the `igbtcal` module are `IcVg`, `IcVg2`, `IcVc`, `IcVc2`, `BV`, `CV`, `Qg`, and `Switching`.
`dtype`	`<k>`	Specifies the datatype of the target. Options are `bool`, `float`, or `int`.
`conditions`	`<d>`	Specifies the condition or split parameters that must be set to calculate the calibration target.
`weight`	`<f>`	Specifies the weight to be applied to the calibration target. Default: `1.0`
`scale`	`<f>`	Sets the scaling factor used to normalize the target value. Default: `1.0`
`range`	`<t>`	Sets the range [low, high] into which the passing target value falls. One of the bound can be `None`.
`abs_value_range`	`<t>`	Sets the range [low, high] into which the absolute value of the passing target falls. One of the bound can be `None`.
`values`	`<t>`	Specifies a list of passing discrete target values.
`dimension`	`<i>`	Specifies the dimension of a high-dimensional (> 2) target.

Argument	Type	Description
error_method	<s>	Specifies the method used to calculate the pointwise error between the simulation response and the reference value. Options are: • "abs": Absolute error: $e_i = \lvert out_i - target_i \rvert$ • "log_abs": Absolute logarithmic error: $e_i = \lvert \log_{10}\lvert out_i \rvert - \log_{10}\lvert target_i \rvert \rvert$ • "log_rel": Relative logarithmic error: $e_i = \left\lvert \frac{\log_{10}\lvert out_i \rvert - \log_{10}\lvert target_i \rvert}{\log_{10}\lvert target_i \rvert} \right\rvert$ • "rel": Relative error: $e_i = \left\lvert \frac{out_i - target_i}{target_i} \right\rvert$ • "signed": Signed error: $e_i = out_i - target_i$ • "signed_relative": Signed relative error: $e_i = \frac{out_i - target_i}{target_i}$ • "square": Squared error: $e_i = (out_i - target_i)^2$ The default applies as follows: • "abs" for type=IcVc or type=IcVc2 or type=Qg or type=Switching • "log_abs" for type=IcVg or type=IcVg2 or type=CV • "rel" for type=None or type=BV

Syntax for xy Profile Targets

```
Target(name=<s>, value=<s>,
       [id=<s>,] [type=<s>,] [dtype=<k>,] [conditions=<d>,]
       [weight=<f>,]
       [simulation_variables=<t>,] [reference_variables=<t>,]
       [simulation_options=<d>,] [reference_options=<d>,]
       [x_range=<t>,] [y_range=<t>,] [x_scale=<f>,] [y_scale=<f>,]
       [error_method=<s>,] [interpolation_options=<d>])
```

Argument	Type	Description
name	<s>	Specifies the name of a Sentaurus Workbench parameter or variable in which the simulation response is available (as a file name).
value	<s>	Specifies the value to be matched by the simulation response. It must be a valid path to a file.
id	<s>	Specifies a unique name or an ID of the target.
type	<s>	Specifies the type for target classification.
dtype	<k>	Specifies the datatype of the target. For files, the only option is str.
conditions	<d>	Specifies the condition or split parameters that must be set to calculate the calibration target.
weight	<f>	Specifies the weight to be applied to the calibration target. Default: 1.0
simulation_variables	<t>	Specifies a tuple with the names of the independent (x) and dependent (y) variables (in that order) in the simulation file.

Argument	Type	Description
`reference_variables`	`<t>`	Specifies a tuple with the names of the independent (x) and dependent (y) variables (in that order) in the reference file.
`simulation_options`	`<d>`	Sets additional options needed to read the values of the simulation file.
`reference_options`	`<d>`	Sets additional options needed to read the values of the reference file.
`x_range`	`<t>`	Sets the range of interest for the independent variable (x) of the reference target xy profile.
`y_range`	`<t>`	Sets the range of interest for the dependent variable (y) of the reference target xy profile.
`x_scale`	`<f>`	Sets the scaling factor for the independent variable (x) of the reference target xy profile.
`y_scale`	`<f>`	Sets the scaling factor for the dependent variable (y) of the reference target xy profile.
`error_method`	`<s>`	Specifies the method used to calculate the pointwise error between the simulation response and the reference value. Options are: • `"abs"`: Absolute error: $e_i = \left\|out_i - target_i\right\|$ • `"log_abs"`: Absolute logarithmic error: $e_i = \left\|\log_{10}\left\|out_i\right\| - \log_{10}\left\|target_i\right\|\right\|$ • `"log_rel"`: Relative logarithmic error: $e_i = \left\|\frac{\log_{10}\left\|out_i\right\| - \log_{10}\left\|target_i\right\|}{\log_{10}\left\|target_i\right\|}\right\|$ • `"rel"`: Relative error: $e_i = \left\|\frac{out_i - target_i}{target_i}\right\|$ • `"signed"`: Signed error: $e_i = out_i - target_i$ • `"signed_relative"`: Signed relative error: $e_i = \frac{out_i - target_i}{target_i}$ • `"square"`: Squared error: $e_i = (out_i - target_i)^2$ The default applies as follows: • `"abs"` for `type=IcVc` or `type=IcVc2` or `type=Qg` or `type=Switching` • `"log_abs"` for `type=IcVg` or `type=IcVg2` or `type=CV` • `"rel"` for `type=None` or `type=BV`
`interpolation_options`	`<d>`	Sets the interpolation options for the simulation data. If simulation data and reference values are in a file but evaluated at different points, then the simulation data is interpolated onto the locations of the target values using the following options: • `"clip"` limits oscillations resulting from interpolation [min, max]. • `"extra"` limits the value taken by the data when extrapolation is necessary [min, max]. • `"log"` determines whether to use logarithmic interpolation (default: `False`). • `"method"` determines the order of the interpolating spline (default: `1`).

Returns

<o>: Target object

Examples

```
Target(name='Error', value=0)

Target(name='t2', id='t_t2', value=10.0, weight=2.0)

Target(name='BottomZ', value=-3.3, conditions={'time': '10'})

Target(name='BTotal_File', id='t_B_SIMS', value='/path/b_sims.plx',
   simulation_variables=['', 'BTotal'], reference_variables=['', 'BTotal'],
   error_method="log_abs", x_range=[0.005, 0.7])

Target(name='IdVgFile', id='t_IdVgLinPlt',
       value='/path/reference_VdLin_nMOS.plt',
       conditions={'Domain': 'NMOS', 'IdVg': 'Lin'},
       simulation_variables=['gate OuterVoltage', 'drain TotalCurrent'],
       reference_variables=['gate OuterVoltage', 'drain TotalCurrent'],
       error_method="log_abs", x_range=[0.0, 1.3],
       interpolation_options={"method": 3})

Target(name='XYBowingCD', id='t_XYBowingCD', value='/path/reference_30.csv',
       conditions={'time': '30'},
       simulation_variables=['Z', 'Radius_Avg'],
       reference_variables=[' Z', ' Radius_Avg'],
       error_method="rel", y_range=[1e-10, 1e30])

Target(name='sIcVc', id="tIcVc", value="IcVc_reference.plt",
    conditions={"IcVg": "0",'IcVc': "1", "BV":"0"},
    reference_variables=["x", "y"],
    simulation_variables=["collector OuterVoltage", "collector TotalCurrent"],
    x_range=[1.0, 3.0],
    type=IcVc)
```

validate_range_ml

This command validates the parameter ranges and adjusts them for subsequent tasks.

Syntax

```
validate_range_ml(parameters=<t>, targets=<t>,
                  [parent=<s>,] [child_dir=<s>,]
                  [task_name=<s>,] [num_retries=<i>,] [constraint=<s>,]
                  [use_checkpoint=<b>,] [reducing_ratio=<f>])
```

Argument	Type	Description
parameters	<t>	Specifies a list of calibration parameter objects.
targets	<t>	Specifies a list of calibration target objects.
parent	<s>	Specifies the parent project. If set to None, then option("parent") is used.
child_dir	<s>	Specifies the directory of the child projects. If set to None, then option("child_dir") is used.
task_name	<s>	Sets the task name. Default: "ML_validate_range"
num_retries	<i>	Sets the number of simulation run retries if there is a simulation runtime failure. Default: 1
constraint	<s>	Specifies an optional constraint or query for an extraction variable of simulations to be included in the training dataset. Default: ""
use_checkpoint	<b>	Specifies whether to use a checkpoint for the machine-learning model as follows: • If set to True, then the machine-learning model and task results are reused if they already exist from a previous run. • If set to False, then the machine-learning model is generated and applied. Checkpoints are files that include the resulting parameter updates and that are written at the end of a machine-learning task. They can be reloaded if available. Default: False
reducing_ratio	<f>	Specifies the ratio used to reduce the parameter range. Default: 0.2

Returns

None

Examples

```
cal.validate_range_ml(parameters=parameters, targets=targets)
```