

Inspect User Guide

Version T-2022.03, March 2022

SYNOPSYS®

Copyright and Proprietary Information Notice

© 2022 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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

Conventions	11
Customer Support	11
<hr/>	
1. Introducing Inspect Functionality	13
Inspect Functionality	13
User Interface	14
Datasets Group Box	15
Curves Group Box	15
Plot Area	15
Supported File Types and Data Formats	16
Data Formats	16
Parameter Files	17
Save Files	17
Starting the Tool From the Command Line	17
Examples of Using Command-Line Options	18
<hr/>	
2. Basic Operations Using the User Interface	20
Loading Datasets	20
Reloading Datasets	20
Updating Datasets Automatically	20
Applying Plotting Actions	21
Zoom Operations	22
Working With Scripts	22
Creating Scripts	22
Running Scripts	23
Preferences	23
Saving Files	24
Exporting Curve Data	24

Contents

Printing Curves Shown in the Plot Area	24
Saving Curves in PNG Format.	25
<hr/>	
3. Working With Curves	26
Selecting Multiple Projects and Groups	26
Creating Curves	26
Automatically Generated Curve Names and Legend Text	27
Selecting and Unselecting Curves	27
Changing the Attributes of a Curve	28
Example: Display a Curve With Data Points and No Lines Between Points	29
Moving the Legend	30
Reordering Curves	30
Creating Curves	30
Creating Curves Automatically When Loading Files	30
Creating Curves Using Formulas	31
Example: Create a Curve That Is the Difference Between Two Curves	32
Creating Curves Using Macros and Library Formulas	32
Deleting Curves	33
<hr/>	
4. Configuring the View	34
Modifying Axes	34
Scaling an Axis	34
Handling Data When Axes Set to Logarithmic Scale	34
Limits of Axes Set to Logarithmic Scale	35
Changing Attributes of Axes	35
Configuring the Plot Area	37
Changing Attributes of Plot Area	37
Adding Labels	39
Displaying the Dataset of a Curve	40
Cleaning Up the Plot Area	40
Sampling Points in the Plot Area	40

5. Curve Interpolation and Operations	42
Curve Operations	42
Limitation of the X-Axis Values of the Dataset	43
More Than One Curve in a Formula	43
Handling Datasets in Formula Processing	44
Dataset Created for Result Curves	45
Curve Handling on Interpolation	46
Determining the Scale (Linear or Logarithmic) of Curves	46
6. Formulas and Macros	47
Using Formulas	47
Formula Library	47
Using Macros	51
Example: ADD Macro	51
Example: DIFFMULT Macro	51
7. Using the Scripting Language	52
Overview of the Scripting Language	52
General-Purpose Commands	53
ft_scalar	53
Reading and Writing Files	53
cv_write	53
fi_writeBitmap	54
fi_writeEps	54
fi_writePs	55
graph_load	55
graph_write	56
param_load	56
param_write	56
proj_getDataSet	57
proj_getList	57
proj_getNodeList	57
proj_load	58
proj_unload	58
proj_write	58

Contents

Creating, Displaying, and Deleting Curves	59
cv_create	59
cv_createDS	59
cv_createFromScript	60
cv_createWithFormula	60
cv_delete	61
cv_display	61
cv_logScale, cv_log10Scale	61
cv_split	62
cv_split_disc	62
Changing Attributes	63
cv_lineColor	63
cv_lineStyle	63
cv_renameCurve	63
cv_set_interpol	64
cv_setCurveAttr	64
gb_setpreferences	65
gr_createLabel	65
gr_deleteLabel	65
gr_formatAxis	66
gr_mappedAxis	66
gr_precision	66
gr_setAxisAttr	67
gr_setGeneralAttr	67
gr_setGridAttr	68
gr_setLegendAttr	68
gr_setLegendPos	69
gr setTitleAttr	69
Accessing Curve Data	69
cv_getVals	69
cv_getValsX	70
cv_getValsY	70
cv_getXaxis	70
cv_getYaxis	71
cv_printVals	71
Transforming Curve Data	71
cv_abs	71
cv_delPts	72

Contents

cv_inv	72
cv_reset	72
Extracting Parameters	73
f_Gamma	73
f_gm	73
f_hideInternalCurves	74
f_IDSS	74
f_KP	74
f_Ron	75
f_Rout.	75
f_showInternalCurves	75
f_TetaG.	76
f_VT	76
f_VT1	77
f_VT2	77
Computing	77
cv_compute	77
cv_getZero	78
macro_define	78
Controlling Scripts	78
script_break	78
script_exit.	79
script_sleep	79
Examples of Using the Scripting Language	79
Computing the Dose of Implanted Arsenic.	79
Creating a Macro to Compute Vt	80

8. Working With Script Libraries	81
Loading Libraries	81
Adding a Site Library	81
Extraction Library	82
cv_linTransCurve	82
cv_scaleCurve	83
ExtractBVi	83
ExtractBVv	84
ExtractEarlyV	84

Contents

ExtractGm	85
ExtractGmb	85
ExtractIoff	85
ExtractMax	86
ExtractRon	86
ExtractSS	87
ExtractValue	87
ExtractVtgm	88
ExtractVtgmb	89
ExtractVti	89
FilterTable	89
Syntax of FilterTable	90
The extend Library	91
cv_addCurve	92
cv_addDataset	93
cv_angularMap	93
cv_autoIncrStyle	94
cv_disp	94
cv_exists	95
cv_getGlobalExtrema	95
cv_getLocalExtrema	96
cv_getNames	96
cv_getRange	97
cv_getXmax	97
cv_getXmin	97
cv_getYmax	98
cv_getYmin	98
cv_integrate	99
cv_isVisible	99
cv_linFit	100
cv_linTrans	100
cv_monotonicX	101
cv_nextColor	101
cv_nextLine	102
cv_nextSymbol	102
cv_resetColor	103
cv_resetFillColor	103
cv_resetLine	104
cv_resetStyle	104

Contents

cv_resetSymbol	105
cv_round	105
cv_scale	106
cv_setFillColor	106
cv_setSymbol	107
cv_sort	107
cv_write	108
dbputs	108
ds_getValue	109
fi_readTxtFile	109
fi_readTxtFileHeader	110
gr_axis	110
gr_resetAxis	111
gr_setStyle	111
ldiff	112
lintersect	112
ltranspose	113
lunion	113
proj_check	114
proj_datasetExists	114
proj_getGroups	115
proj_groupExists	115
proj_loadPlx	116
The PhysicalConstants Library	117
IC-CAP Model Parameter Extraction Library	118
Exporting Data	119
Header Information	119
userInput	119
iccapInput	119
output	120
Array Data	121
Curve Comparison Library	121
cvcmp_CompareTwoCurves	121
cvcmp_DeltaTwoCurves	122
References	122

Contents

A. Elements of User Interface	123
Toolbar Buttons	123
File Menu	124
Edit Menu	125
Curve Menu	126
Script Menu	127
Extensions Menu	128
Help Menu	128

B. Known Limitations	129
The diff(...formula...) and integr(...formula...) Operators	129
The vecvalx(...formula...) and vecvaly(...formula...) Operators	129
No Support for Right Y-Axes	130

About This Guide

The Inspect tool is a curve display and analysis program. It works with curves specified at discrete points. Inspect allows users to work interactively with data using both a graphical user interface and a scripting language.

For additional information, see:

- The TCAD Sentaurus™ release notes, available on the Synopsys® SolvNetPlus support site (see [Accessing SolvNetPlus on page 12](#))
- Documentation available on the SolvNetPlus support site

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.
<code>Courier font</code>	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.
<i>Italicized text</i>	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.
Key+Key	Indicates keyboard actions, for example, Ctrl+I (press the I key while pressing the Control key).
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 customer support website 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

1

Introducing Inspect Functionality

This chapter introduces the Inspect tool and its user interface.

Inspect Functionality

The Inspect tool can display and analyze curves. It features a user interface, a scripting language, and an interactive language for computations with curves.

An Inspect curve is a sequence of points defined by a one-dimensional array of x-coordinates and y-coordinates (floating-point values). An array of coordinates that can be mapped to one of the axes is referred to as a *dataset*. In Inspect, datasets can be combined and mapped to the x-axis and y-axis to create and display a curve.

The Inspect tool works on data consisting of *groups* of datasets. Each group consists of two or more datasets of equal length, where elements with an identical index represent related values. Datasets in a group can be divided into *named groups*. By pairing related values of two datasets from the same group, points (or nodes) of a discrete curve are obtained.

Usually, a dataset represents a physical quantity, such as voltage, current, or time. Groups of datasets represent functionally related physical quantities, for example, measurements of current and voltage, and the times at which these measurements are taken. Named groups represent semantically related datasets, for example, meshing results at one of several contacts of a semiconductor device.

In addition to its name, a dataset can have other attributes associated with it, for example, the name of the physical quantity represented by the dataset, the name of the unit in which this quantity has been measured, the preferred color of the curve when it is visualized, and interpolating function. Depending on the particular input file format, this information can be stored partially in the data file and partially in separate files.

The Inspect tool can read different data formats and file formats (see [Data Formats on page 16](#)).

Data in TDR and TIF formats contains the names of the physical quantities that datasets represent. The TDR and TIF formats allow you to split a dataset group into named groups.

Chapter 1: Introducing Inspect Functionality

User Interface

Data in XGRAPH format always has groups consisting of two datasets only. Additional dataset attributes are specified inside the file with appropriate keywords.

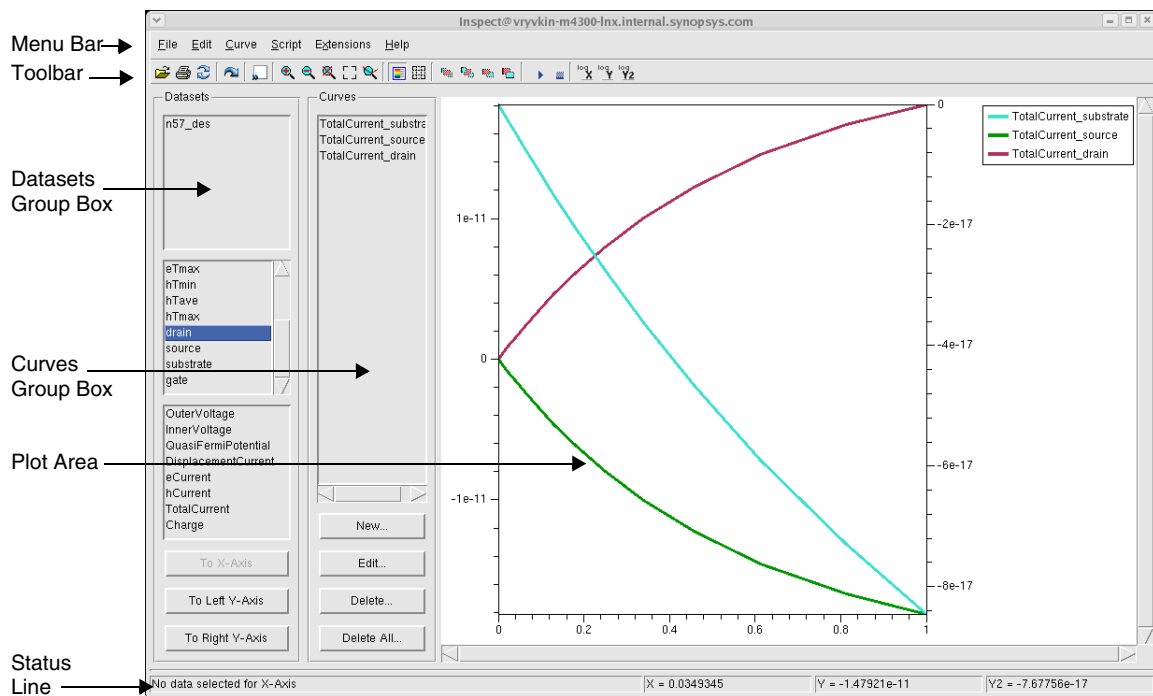
To distinguish datasets from different data files, the datasets from one data file are grouped into a *project*. The name of the data file without an extension is taken as the project name. When more than one file with the same name is loaded, Inspect adds the suffix *.n* to the project name, where *n* is the smallest number not yet used as a suffix for another project name.

User Interface

The user interface of the Inspect tool has the following work areas (see [Figure 1](#)):

- Datasets group box
- Curves group box
- Plot area

Figure 1 Main window showing work areas



Menus and toolbar buttons are documented in [Appendix A on page 123](#).

The status line at the bottom of the main window displays information about the current Inspect session and the position of the pointer in the plot area.

Datasets Group Box

This group box has the following panes for selecting and combining datasets to create curves:

- The *top pane* lists the currently loaded data files (or *projects*). In the example shown in [Figure 1 on page 14](#), only one data file `n57_des` has been loaded (the file extension is not displayed).
- The *middle pane* lists the names of the dataset groups belonging to the selected data file. A group having one or more datasets can correspond to an electrode or a thermode of a device. Datasets that do not belong to any group are also displayed. In [Figure 1](#), `hTave` and `hTmax` are independent datasets. In addition, `drain`, `source`, `substrate`, and `gate` are groups, each having several datasets.
- The *bottom pane* lists the names of the datasets belonging to the selected group.

The **To X-Axis**, **To Left Y-Axis**, and **To Right Y-Axis** buttons map datasets to a particular axis.

Curves Group Box

This group box has one pane that displays the names of existing curves, and it has the following buttons:

- Click **New** to create a curve using the formula library (see [Creating Curves Using Formulas on page 31](#)).
- Click **Edit** to change the graphical attributes of a curve (see [Changing the Attributes of a Curve on page 28](#)).
- Click **Delete** to remove curves that are selected in the pane.
- Click **Delete All** to remove all curves in the pane.

Plot Area

The plot area is where curves are drawn. The toolbar buttons are used to change the coordinate system for zooming sessions, to display or remove the legend text, to change the order in which curves are displayed, and to switch between linear and logarithmic scale.

Supported File Types and Data Formats

This section describes the supported file types and data formats.

Data Formats

Inspect works with different file formats, which contain a series of points, described by x-coordinates and y-coordinates, representing datasets. Inspect handles and displays these datasets as curves.

Table 1 *Supported data formats*

Data format	Description
CSV	Comma-separated value format, which is recognized by many applications. The file extension is <code>.csv</code> .
PLT	Plain text format for 1D curves.
PLX	Format generated by Sentaurus Process.
TDR	Format recognized by most Synopsys TCAD tools. The file extension is <code>.tldr</code> .
TIF	Synopsys TCAD format for I–V curves, recognized by most Synopsys TCAD tools. The file extension is <code>.iv1</code> . For a description of the TIF format, see <i>Taurus™ Visual User Guide</i> , Data Formats.
TXT	Tab-delimited text format. The file extension is <code>.txt</code> . Each curve is written into a block with the curve name, and the x- and y-data.
XGRAPH	Each curve is written into a block with the curve name, and the x- and y-data. The file extension is <code>.xy</code> . Typically, there is one xy data–point pair per line. Each value or column is separated by space, tabs, commas, semicolons, or colons. For more information, go to http://www.xgraph.org .
XMGR	Format used by the shareware xy plotting tool Xmgr. The file extension is <code>.xmgr</code> . For more information, go to https://plasma-gate.weizmann.ac.il/Xmgr/ .

Parameter Files

When the Inspect tool is customized interactively according to user preferences, the current setup can be stored in a parameter file, which usually has the extension `.par`. In a parameter file, Inspect stores the following information:

- Plot area attributes
- Coordinate area attributes
- Axes attributes
- User-defined macros
- Printer setup
- Curve attributes

When the Inspect tool starts, it looks for a parameter file named `inspect.par` in the current directory. If such a file is found, then the tool loads it and sets the plot settings, macros, and printer setup according to the values in this file. If no file is found in the current directory, then the tool looks for a parameter file named `inspect.par` in the `STDB` directory. If no file is found, then the tool uses the default values.

You can also load a parameter file explicitly when starting the tool or during its execution.

Curve attributes saved in the parameter file are not applied automatically to curves in the tool. To do this, you must specify the command-line option `-applyCurveAttr` when starting Inspect.

Save Files

The entire current state of Inspect (projects, curves, and settings) can be saved to a save file, which is used to restore the saved state at any time and usually has the `.sav` extension.

Data from all loaded projects is also stored in a save file. This means that for restoring the saved state, the data files are no longer necessary.

Starting the Tool From the Command Line

You can start the Inspect tool from the command line by typing:

```
inspect [<options>] [<FILES>]
```

You can also specify options (see [Table 2 on page 18](#)) and files on the command line. See [Examples of Using Command-Line Options on page 18](#).

Chapter 1: Introducing Inspect Functionality

Starting the Tool From the Command Line

The <FILES> arguments can be one or several data files, or a save file that is loaded when Inspect is started. Inspect automatically distinguishes between data files and save files.

Note:

You can also start Inspect from Sentaurus Workbench.

Table 2 *Command-line options*

Option	Description
-applyCurveAttr	Applies curve attributes saved in a parameter file (.par) and a save file (.sav).
-batch	If specified, then the Inspect tool does not open the user interface while executing a script file.
-c FILE ¹	Reads the specified setup file after Inspect is launched.
-display	Sets the display to use.
-f FILE ¹	Executes the specified script file after Inspect is launched.
-geometry	Sets the size and position of the main window of the Inspect tool.
-h[elp]	Displays information about the command-line options.
-m FILE ¹	Executes the specified macro file after Inspect is launched.
-oldInterpol	Forces the Inspect tool to use the interpolation criteria of earlier versions.
-oldplx	Loads a .plx file and automatically creates curves with the old scheme.
-v[ersion]	Prints the version of the Inspect tool.
-verbose	Displays all messages.

1. *Simulation results (.plt, .tdr, .ivl) files as well as save (.sav) files.*

Examples of Using Command-Line Options

The Inspect tool starts in interactive mode and loads datasets from the n53_des.plt file:

```
inspect n53_des.plt
```

Inspect starts in interactive mode, loads three data files, and reads the plot area, axes, macros, and printer setup from the parameter file mySetup.par:

```
inspect file1.plt file2.plt file3.plt -c mySetup.par
```

Chapter 1: Introducing Inspect Functionality

Starting the Tool From the Command Line

The Inspect tool starts in interactive mode and executes the `bipolar.cmd` script file:

```
inspect -f bipolar.cmd
```

The Inspect tool starts in batch mode and executes the `bipolar.cmd` script file:

```
inspect -batch -f bipolar.cmd
```

2

Basic Operations Using the User Interface

This chapter describes the basic operations using the Inspect user interface.

Loading Datasets

You must open a data file to load a dataset.

To load a dataset:

1. Choose **File > Load Dataset**, or press Ctrl+L.

The Load Dataset dialog box opens.

2. Select a data file by changing the **Files of type** field if needed.
3. Click **Open** or double-click a file to open it.

Reloading Datasets

You can reload displayed datasets if the file is updated while viewing it in the Inspect tool.

To reload a dataset:

- Choose **File > Update Datasets**, or press Ctrl+U.

Updating Datasets Automatically

If a dataset displayed in the plot area is updated frequently by other tools, then it can be useful to reload the dataset frequently so that refreshed data is shown in the plot area.

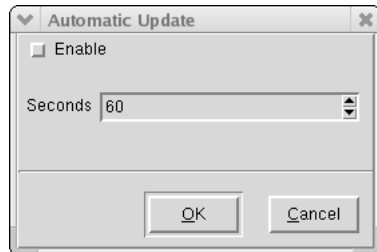
Chapter 2: Basic Operations Using the User Interface

Applying Plotting Actions

To update datasets automatically:

1. Choose **File > Automatically Update Datasets**.

The Automatic Update dialog box opens.



2. Select **Enable**.
3. Set the time (in seconds) between reloads.
4. Click **OK**.

Applying Plotting Actions

When working with multiple datasets of a similar structure, it is useful to apply several plotting actions made on one dataset to other datasets. These plotting actions include creating curves (explicitly and by formula) and changing curve attributes. Inspect stores the set of actions made on the current dataset.

To repeat plotting actions for another dataset:

1. Select the dataset or datasets.
2. Choose **Edit > Redo Last Plot**, or press Ctrl+E.

All plotting actions are applied to the selected datasets.

This feature is helpful when datasets contain data of a similar structure, for example, projects with groups of the same names. Otherwise, it can happen that the current dataset does not contain some data necessary for creating a curve. In this case, Inspect generates an error message.

Note:

The Inspect tool stores plotting actions applied to the currently selected dataset only. When you select another dataset, the accumulated set of stored actions is cleaned up.

Zoom Operations

To perform zoom operations on curves displayed in the plot area, use the relevant toolbar buttons (see [Table 7 on page 123](#)).

Working With Scripts

In Inspect, any sequence of operations can be stored and reproduced using scripts (see [Chapter 7 on page 52](#)). The recorded operations are repeated when the script is executed. The following operations can be recorded in a script:

- Loading and unloading projects
 - Loading and saving the current project
 - Changing axis attributes
 - Exporting, creating, and deleting curves
 - Changing curve attributes
 - Transforming actions on curves
 - Any use of the formula library to create other curves
-

Creating Scripts

To create a script:

1. Choose **Script > Record > Start**.

The Record Script File dialog box opens.

2. Select or create a script file, and click **Save**.

Inspect starts to store every operation until you stop recording.

3. Choose **Script > Record > Stop**.

Running Scripts

To run a script:

1. Choose **Script > Run Script**, or press Ctrl+R.

The Run Script File dialog box opens.

2. Select a script file.
3. Click **Open**.

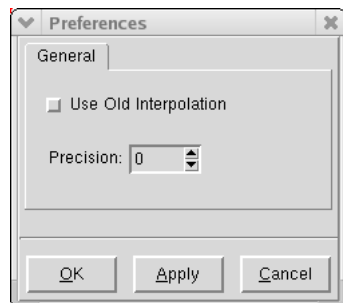
Preferences

In Inspect, preferences relate to the precision of values handled for curve coordinates and the interpolation method used to operate on and display curves.

To set the preferences:

1. Choose **File > Preferences**.

The Preferences dialog box opens.



2. In the **Precision** box, select the number that indicates how many decimal digits are used to handle coordinates for curve points.
3. If required, select **Use Old Interpolation** to force Inspect to use the interpolation criteria of earlier versions to handle all computations with curves.
4. Click **OK**.

Saving Files

To save files, you can either:

- Choose **File > Save Setup** to save the current setup in a parameter `.par` file.
- Choose **File > Save All** to save the entire current state of the Inspect tool in a `.sav` file.

Exporting Curve Data

Curve data can be written to data files in different formats (see [Data Formats on page 16](#)). You can select different formats from the **File > Export** command.

To export curve data in TDR format:

1. Choose **File > Export > TDR**.
The Write `.tdr` File dialog box opens.
2. Select or create a TDR file.
3. Click **Save**.

Printing Curves Shown in the Plot Area

Curves shown in the plot area can be printed as a single image.

To print curves shown in the plot area:

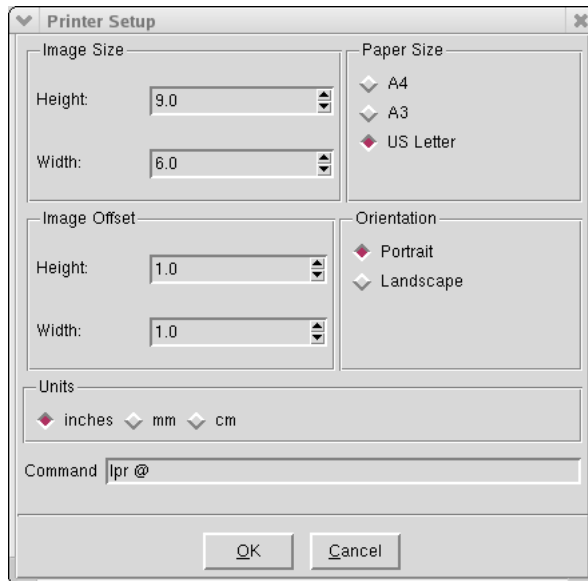
1. Choose **File > Print**, or press Ctrl+P.
The Printer Setup dialog box opens.
On the Windows operating system, a standard print dialog box is displayed. On Linux operating systems, a special print dialog box opens.
2. Specify the print configuration as required.

Note:

In the **Command** field, you can specify a command for using the printer.

Chapter 2: Basic Operations Using the User Interface

Saving Curves in PNG Format



3. Click **OK**.

Saving Curves in PNG Format

Curves shown in the plot area can be saved as a bitmap file in PNG format. This bitmap file can be imported into applications for documentation and reporting purposes.

To save curves in PNG file format:

1. Choose **File > Write Bitmap**, or press Ctrl+W.

The Write PNG bitmap file dialog box opens.

2. Enter the name of the file.
3. Click **Save**.

You can save the curves shown in the plot area in a `.png` file directly from your Inspect script using the `fi_writeBitmap` command (see [fi_writeBitmap on page 54](#)). This is a screen-image exporting method that works when Inspect starts in interactive mode.

Note:

Since this exporting method is based on the X11 utility `xwd`, it works only if a valid `DISPLAY` is available, that is, an X server is required.

3

Working With Curves

This chapter describes how to create and work with curves using the Inspect user interface.

Selecting Multiple Projects and Groups

The Inspect tool displays curves that are formed by datasets from different data files.

You select projects and groups from the Datasets group box (see [Datasets Group Box on page 15](#)). If you select more than one group, only the dataset names that exist in all the selected groups appear in the bottom pane.

In [Figure 1 on page 14](#), if you select two groups (`drain` and `source`) in the middle pane, then the `OuterVoltage`, `InnerVoltage`, `QuasiFermiPotential`, `DisplacementCurrent`, `eCurrent`, and `hCurrent` dataset names appear in the bottom pane. If you select the group `hTmax`, in addition, no dataset names are displayed in the bottom pane because no datasets with identical names exist in all the selected groups.

Note:

The names of datasets must be identical. Datasets, themselves, can be different.

When more than one project is loaded, the same rule applies. Consequently, Inspect shows only common groups and datasets of multiple-selected projects in the middle and bottom panes of the Datasets group box.

Creating Curves

The first curve displayed in [Figure 1 on page 14](#) maps `time` to the x-axis and `TotalCurrent` of `drain` to the left y-axis.

To create a curve:

1. Select the dataset group `time` in the middle pane of the Datasets group box.
2. Click **To X-Axis**.

Chapter 3: Working With Curves

Automatically Generated Curve Names and Legend Text

3. Select the group `drain` in the middle pane of the Datasets group box.
4. Select the dataset `TotalCurrent` from the bottom pane of the Datasets group box.
5. Click **To Left Y-Axis**.

The Inspect tool draws a curve in the plot area using these two datasets. The name of the curve is generated automatically using names and other attributes of groups and datasets (and possibly projects). The upper-right corner of the plot area shows the legend, which displays curve names and drawing styles.

The second curve is created in the same way: `time` is mapped to the x-axis and `TotalCurrent` from `source` is mapped to the left y-axis. The next curve is created with `time` mapped to the x-axis and `TotalCurrent` from `substrate` mapped to the right y-axis.

When these steps are completed, the main window resembles [Figure 1 on page 14](#).

Automatically Generated Curve Names and Legend Text

When a curve is created, a default name for it is generated. With plot files (TDR or TIF files), the name is a combination of the physical quantity name and the group name of the y-dataset if the dataset belongs to a group.

For example, if the physical quantity name is `OuterVoltage` and the dataset belongs to a group named `drain`, the curve name is `OuterVoltage_drain`.

With XGRAPH files, a default name is created using the data file name and the comment line preceding the dataset pair in the file. If the generated curve name already exists, an `.n` suffix is added, where `n` is the smallest number not yet used as a suffix in another name.

In addition to a name, a curve has legend text, which identifies the curve in the plot area. When a curve is created, the corresponding text is initialized with the curve name.

Selecting and Unselecting Curves

To select a curve, click the curve name in the Curves group box, or click the curve name in the legend.

After selection, the curve name is highlighted in the Curves group box, the plot area, and the legend.

To unselect a curve, right-click in the plot area.

Changing the Attributes of a Curve

A curve is displayed according to its attributes. When a curve is created, default values are assigned to all attributes. A curve is drawn with lines connecting nodes and, optionally, with node markers.

To change the attributes of a curve:

1. Double-click a curve in the plot area or in the Curves group box.

The Curve Attributes dialog box opens (see [Figure 2 on page 29](#)).

2. On the **General** tab, change the axis to which the curve is mapped, if required.
3. On the **Line** tab, change the following attributes of lines:

Color Default color is assigned from a list of colors that are not assigned to existing curves.

Style When all available colors are exhausted, a line drawing style is assigned from a list of styles, so that each curve has a unique combination of color and style.

Width Line width in pixels. Default: 1

4. On the **Marker** tab, change the following attributes of node markers:

Shape No node markers by default.

Size Size of markers in pixels. Default: 5

Outline Color It is the same as the line color by default.

Outline Width Width of markers in pixels. Default: 1

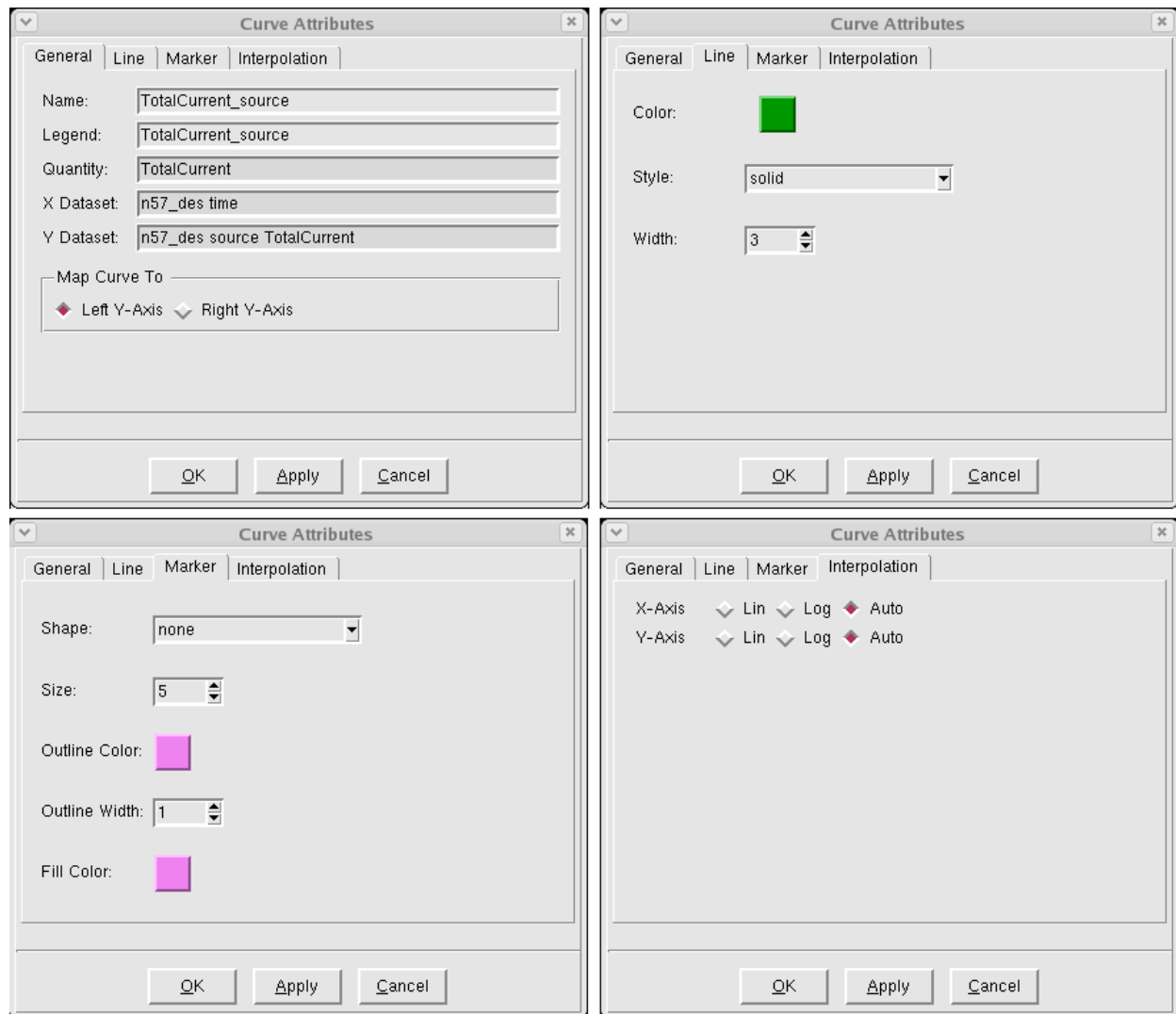
Fill Color It is the same as the line color by default.

5. On the **Interpolation** tab, change the interpolation used on the x-axis and y-axis, if required.
6. Click **Apply**.
7. Click **OK**.

Chapter 3: Working With Curves

Changing the Attributes of a Curve

Figure 2 Curve Attributes dialog box with all tabs shown



Example: Display a Curve With Data Points and No Lines Between Points

To display a curve with data points and no lines between these points:

1. Double-click a curve in the plot area or in the Curves group box.

The Curve Attributes dialog box opens (see [Figure 2](#)).

2. Click the **Marker** tab.

Chapter 3: Working With Curves

Moving the Legend

3. In the **Shape** box, select **circle**.
4. Click the **Line** tab.
5. Set **Width** to 0.
6. Click **Apply**.
7. Click **OK**.

Moving the Legend





To move the legend in the plot area:

- Use the middle mouse button and drag the legend to its new location.

Reordering Curves

You can change the order in which curves are displayed in the plot area to distinguish one curve from another, for example, if some have a significant intersection.

To reorder curves, choose one of the following options from the **Curve** menu (see [Curve Menu on page 126](#)) or click the corresponding toolbar button:

- **Drawing Order > Move to Front** 
- **Drawing Order > Move to Back** 
- **Drawing Order > Move Forward** 
- **Drawing Order > Move Backward** 

Creating Curves

You can create curves in different ways.

Creating Curves Automatically When Loading Files

When a PLT, TDR, or TIF file is loaded, the Inspect tool does not create curves immediately in the plot area. This is because a curve might be formed from any pair of datasets belonging to different groups and only a few of these are likely to interest users. Curves must be created explicitly, when required, by mapping pairs of datasets to x-axes and y-axes.

Chapter 3: Working With Curves

Creating Curves

When a PLX or an XGRAPH file is loaded, Inspect creates curves automatically in the plot area, from the pairs of datasets defined in the loaded file, with the y-dataset being mapped to the left y-axis. The y-dataset can be remapped later to the right y-axis.

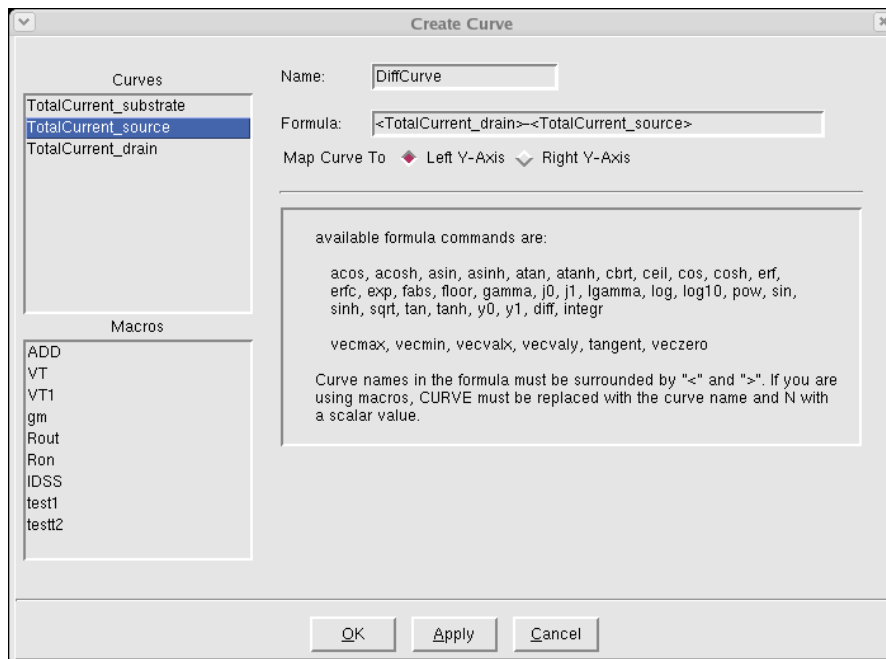
Creating Curves Using Formulas

You can create curves by applying formulas to existing curves (see [Using Formulas on page 47](#)).

To create a curve using a formula:

1. In the Curves group box, click **New**.

The Create Curve dialog box opens.



2. In the Curves pane, click a curve to highlight it.
3. In the **Name** field, enter the name of the new curve.
The Inspect tool provides a default.
4. In the **Formula** field, enter the formula to be used to create the curve.
5. Map the curve to either the left y-axis or the right y-axis.
6. Click **Apply**.
7. Click **OK**.

Example: Create a Curve That Is the Difference Between Two Curves

To create a curve that is the difference between two curves:

1. In the Curves group box, click **New**.

The Create Curve dialog box opens.

2. In the Curves pane, click a curve to highlight it.

The curve name appears in the **Formula** field enclosed by angle brackets.

3. Type a hyphen after the closing angle bracket in the **Formula** field.

4. Double-click a different curve in the Curves pane.

The name of the second curve appears in the **Formula** field.

5. Select the axis to which the new curve is mapped.

6. Click **Apply**.

7. Click **OK**.

The new curve is displayed in the plot area.

Creating Curves Using Macros and Library Formulas

You can create and handle new curves using macros and library formulas included in the Inspect tool. You can create macros using the Macro Editor, which allows you to select existing macro functions, different operations, and formulas from the libraries (see [Figure 3 on page 33](#)). For more information about using macros, see [Using Macros on page 51](#).

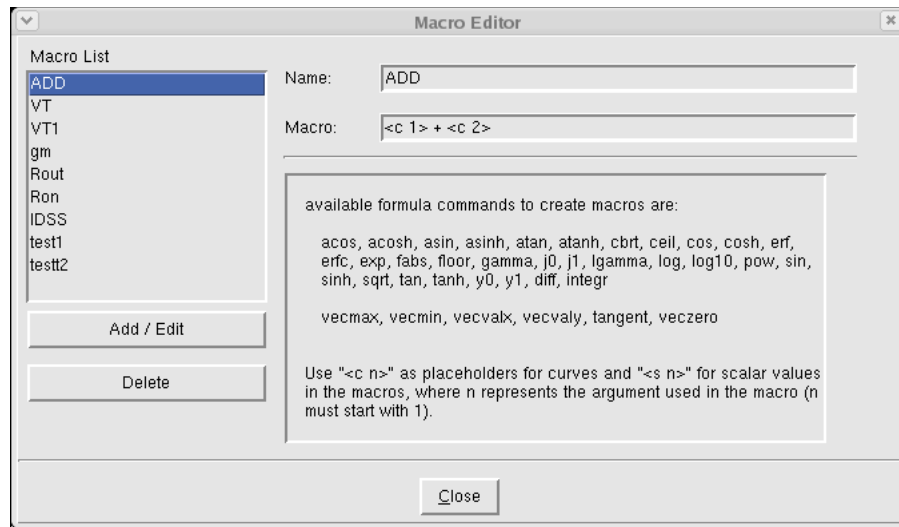
Macros are stored in the file `inspect_macro.par` in the `STDB` directory. This file is created automatically the first time the Inspect tool is run. Initially, it stores predefined macros. You can then add or modify macros using the Macro Editor.

To open the Macro Editor, choose **Edit > Define Macros**.

Chapter 3: Working With Curves

Deleting Curves

Figure 3 Macro Editor



Deleting Curves

You can delete either selected curves or all curves listed in the Curves group box.

To delete selected curves:

1. Select the curves in the Curves group box.
2. Click **Delete**.
3. Confirm the deletion.

To delete all curves:

1. Click **Delete All**.
2. Confirm the deletion.

4

Configuring the View

This chapter describes how to modify the way in which Inspect displays loaded datasets.




Modifying Axes

This section describes different ways you can modify the axes.

Scaling an Axis

You can scale the x-axis, the left y-axis, and the right y-axis. The scale of each axis can be changed independently.

To change the scale of an axis:

1. Click the relevant toolbar button:
 - The  button switches on logarithmic scale on the x-axis.
 - The  button switches on logarithmic scale on the left y-axis.
 - The  button switches on logarithmic scale on the right y-axis.
2. Click the corresponding toolbar button again to switch off logarithmic scale for an axis.

When logarithmic scale is switched off, the axis reverts to linear scale.

Handling Data When Axes Set to Logarithmic Scale

If an axis is set to logarithmic scale, Inspect handles data in the following way:

- Negative values are set to positive.
- Zero values are set to 1e-20.

When an axis switches to linear scale, data is restored to its original values.

Limits of Axes Set to Logarithmic Scale

When an axis is set to linear scale, you can set the minimum and maximum values of the axis.

When an axis is set to logarithmic scale, Inspect might be unable to set an axis to the given values. In this case, Inspect sets the minimum and maximum values of the axis to the nearest power of 10 values.

Changing Attributes of Axes

To change the attributes of axes:

1. Choose **Edit > Axes**, or press Ctrl+A, or double-click any axis in the plot area.

The Axes dialog box opens (see [Figure 4 on page 36](#)).

2. Click the required tabs to change attributes.
3. Click **OK** or **Apply** to accept the changes.

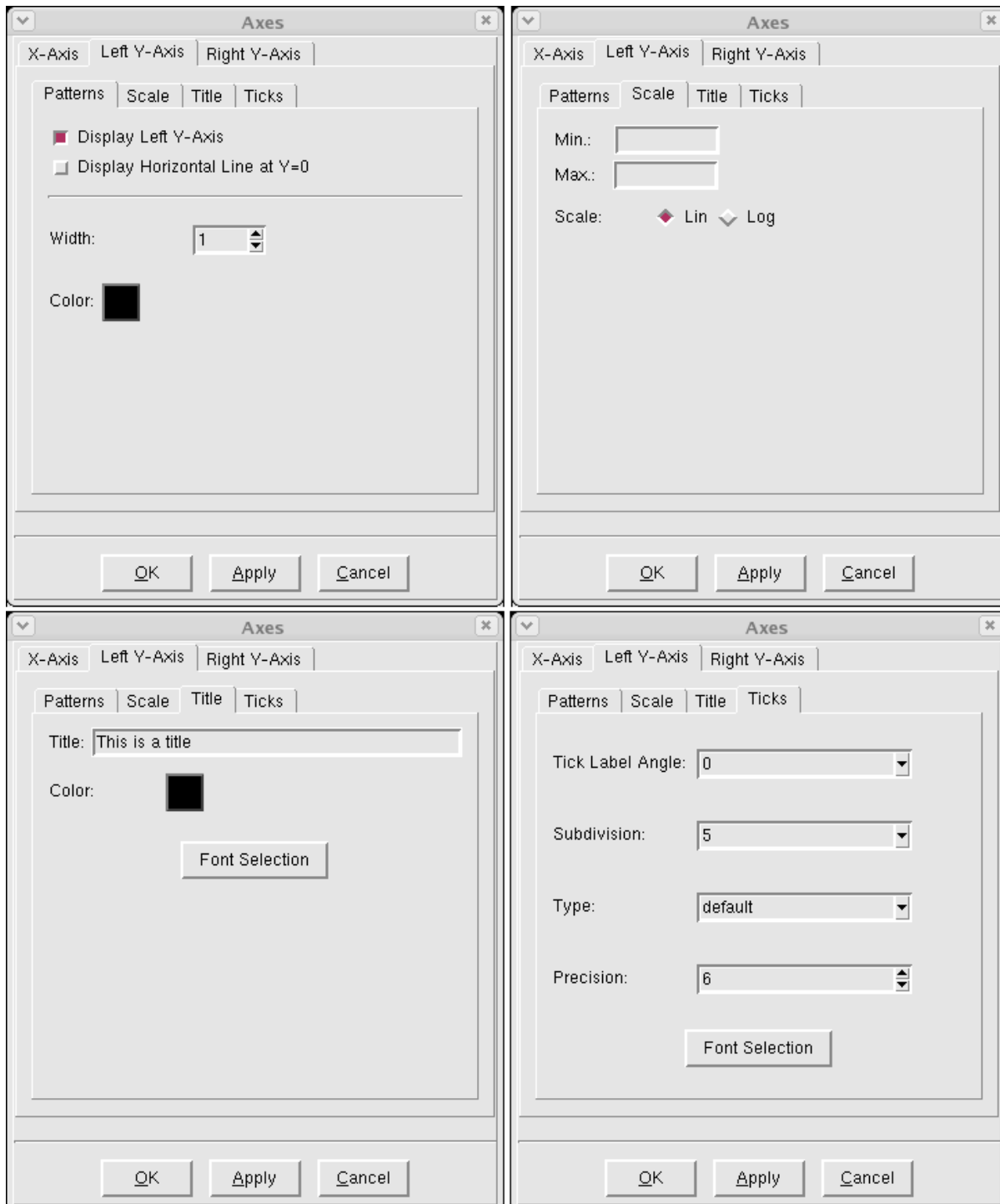
For example, to change the x-axis from linear scale to logarithmic scale:

1. Click the **X-Axis** tab.
2. Click the **Scale** tab.
3. Select **Log**.
4. Click **OK** or **Apply** to accept the changes.

Chapter 4: Configuring the View

Modifying Axes

Figure 4 Axes dialog box



Configuring the Plot Area

This section describes different ways you can modify the plot area.

Changing Attributes of Plot Area

You can change the appearance of the plot area and modify attributes such as the name of the plot area, the legend text that references the displayed curves, the plot frame, and the grid.

To change the attributes of the plot area:

1. Choose **Edit > Plot Area**, or press Ctrl+G.

The Plot Area dialog box opens (see [Figure 5 on page 38](#)).

2. Click the required tabs to change attributes.
3. Click **OK** or **Apply** to accept the changes.

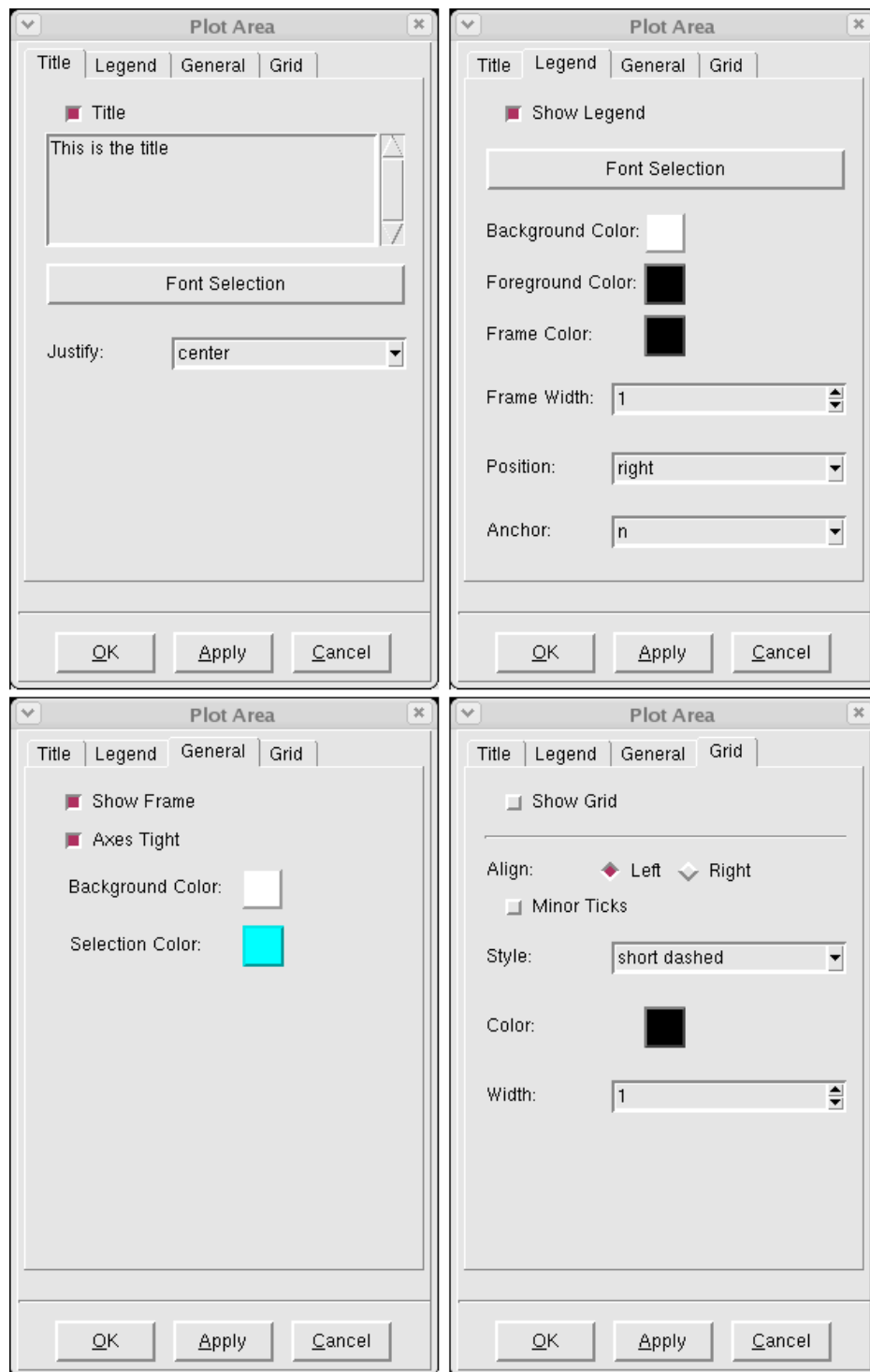
For example, to change the position of the legend text in the plot area:

1. Click the **Legend** tab.
2. In the **Position** box, select the new position.
3. Click **OK** or **Apply** to accept the changes.

Chapter 4: Configuring the View

Configuring the Plot Area

Figure 5 *Plot Area dialog box showing all tabs*



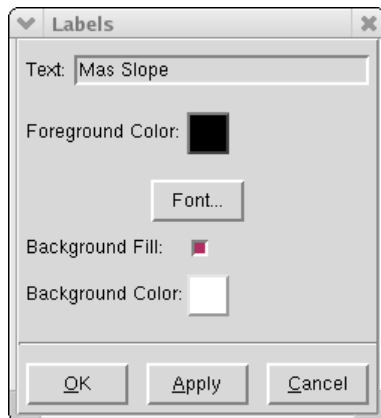
Adding Labels

You can add labels to the plot area. These labels provide useful information about the mapped curves. Labels can be edited and removed from the plot area.

To add a label:

1. Choose **Edit > Labels > Add**.

The Labels dialog box opens.



2. In the **Text** field, type the label text.
3. Select a color and font for the label.
4. Click **OK** or **Apply** to insert the new label in the plot area.

To move a label:

1. Select a label in the plot area using the middle mouse button.
2. Move the label inside the plot area as required and release the middle mouse button.

To delete a label:

1. Choose **Edit > Labels > Remove**.

The pointer changes to the delete mode.

2. In the plot area, click the label to be removed.

When the label is removed, the pointer reverts to the standard mode.

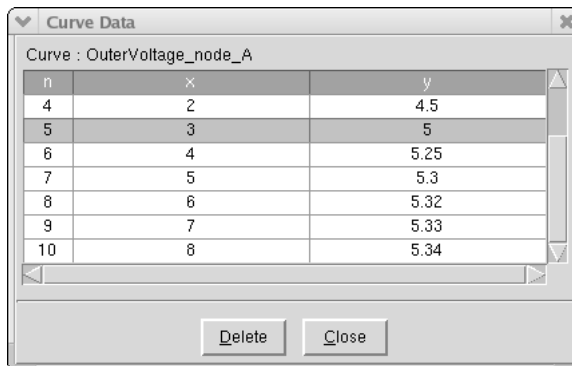
Displaying the Dataset of a Curve

Each curve displayed in the plot area has an associated dataset.

To view the points (data) included in the dataset of a specific curve:

1. Select a curve from the Curves group box.
2. Choose **Curve > Curve Data**, or press Ctrl+D.

The Curve Data dialog box opens, showing a table of the x-coordinates and y-coordinates for each point in the datasets represented by the selected curve.



3. Highlight a data point in the table.
4. Click **Delete** to remove it from the displayed curve.

Cleaning Up the Plot Area

You can clean up the plot area, in which case, all existing curves are deleted, the legend is removed, and the plot area is reinitialized.

To clean up the plot area:

- Choose **Edit > Clean Plot Area**, or click the  toolbar button.

Sampling Points in the Plot Area

To sample points in the plot area:

1. Choose **Curve > Inspector**.

The Inspector dialog box opens (see [Figure 6](#)).

2. In the plot area, select the first point by clicking a specific location (usually on a curve).

Chapter 4: Configuring the View

Configuring the Plot Area

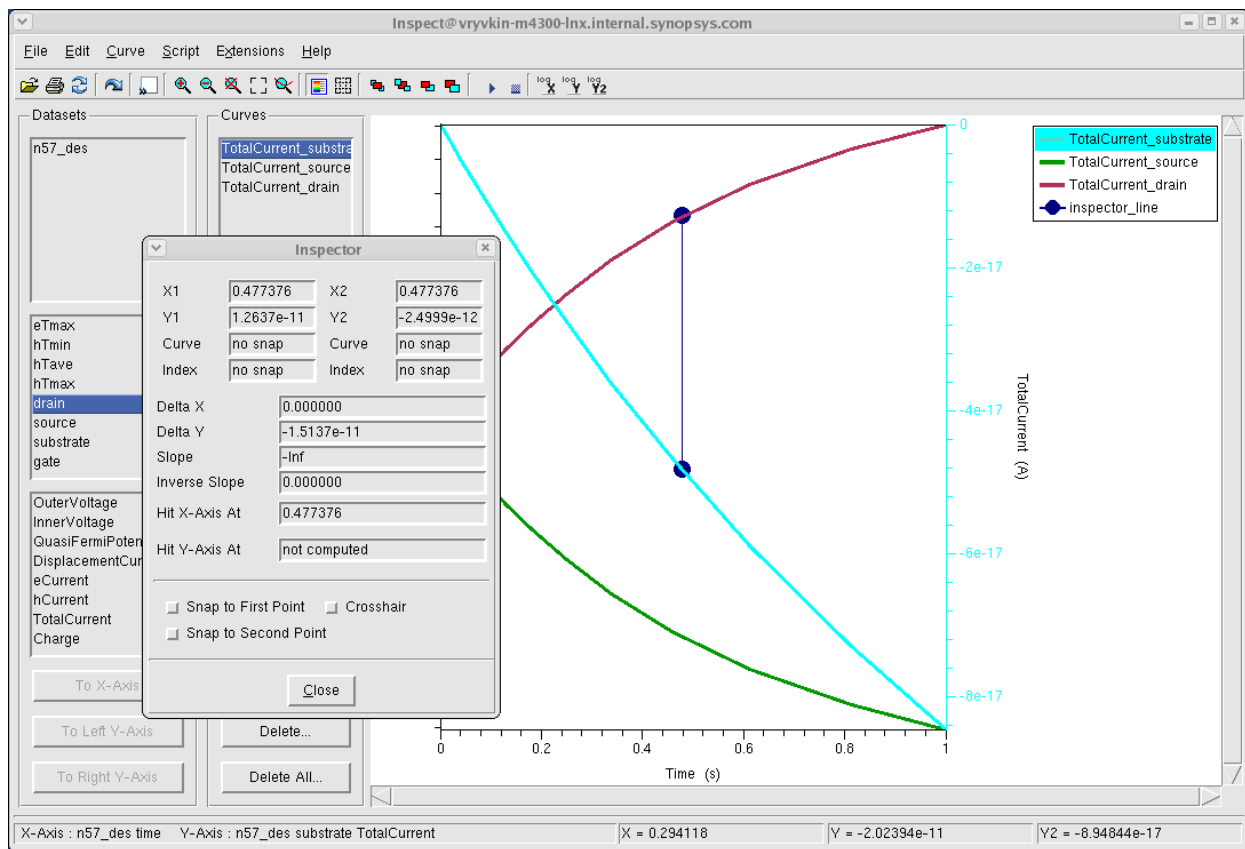
3. Drag the pointer to a second location (usually on another curve) to mark the second point.

Note:

The Inspector dialog box works only for the x-axis and left y-axis.

Positions are represented by circles that are connected by a line. The Inspector dialog box shows different values calculated from the two selected positions.

Figure 6 Example of using Inspector dialog box to sample points in the plot area



5

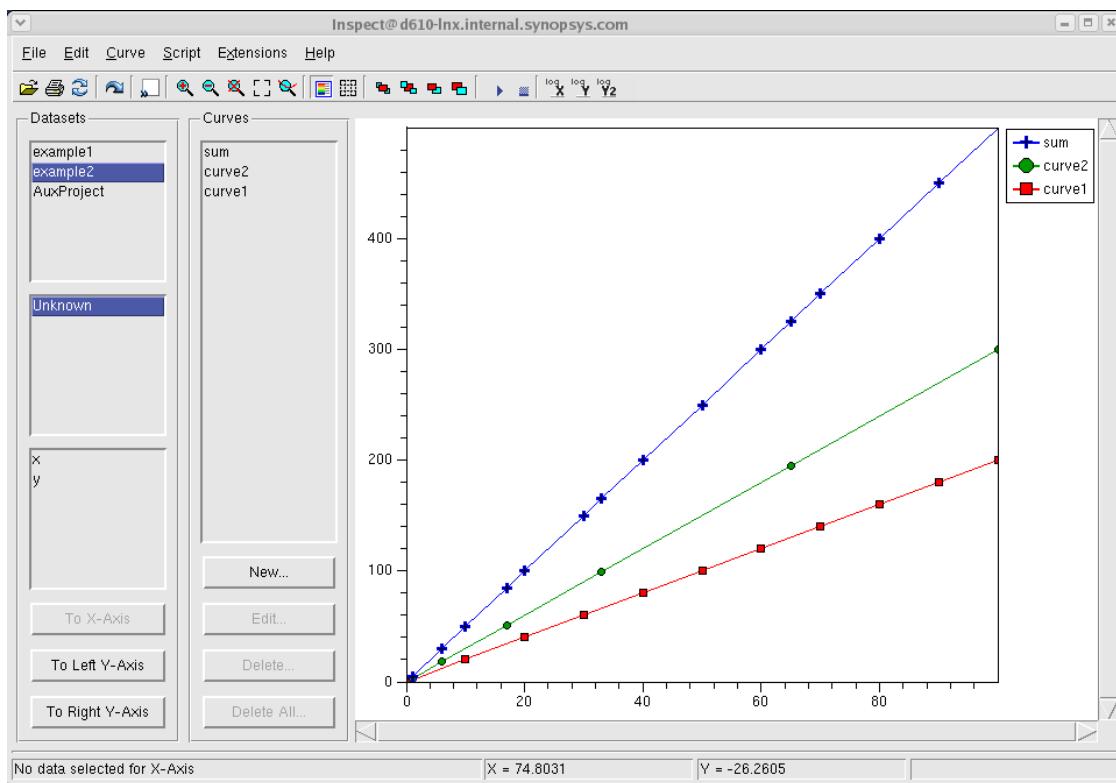
Curve Interpolation and Operations

This chapter discusses curve interpolation and operations.

Curve Operations

A curve is defined as a set of two or more (x, y) points. Each curve has its own set of points called datasets. Inspect can display the resulting (continuous) curve by plotting all data points and completing the curve with a graphical linear interpolation method. [Figure 7](#) shows three different curves, each defined by different datasets.

Figure 7 Different curves displayed as lines



The Inspect tool offers different operations with curves. This requires an efficient way to handle curves and to create datasets for curves resulting from operations with other curves.

Some operations result in a new curve or a scalar value. These operations include the sum of two or more curves, integration, differentiation, and the maximum value on the y-axis or the x-axis.

Limitation of the X-Axis Values of the Dataset

Note:

Only curve segments that have x-axis values monotonically increasing or decreasing can be used in a formula.

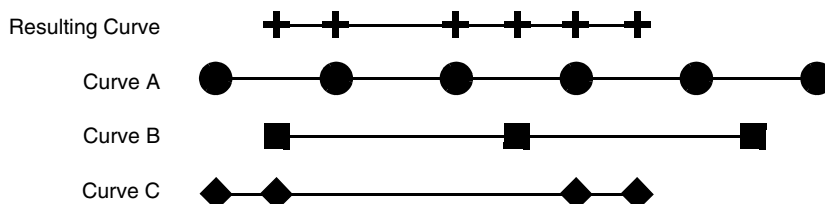
Curves used in a formula are checked for monotonically increasing or decreasing values inside the range defined by the current zoom level in the plot area. This means formulas apply only to the currently displayed points. By defining the optimal zoom level for the selected curves, it is possible to cut off curve segments that do not have monotonically increasing or decreasing x-values.

This general rule has *one exception*. An Inspect formula can involve only one curve with nonmonotonous x-axis values. In this case, Inspect splits this curve into monotonous segments of x-axis values, applies the formula to those segments, and builds the resulting curve automatically. For example, a scaling formula can be applied to a curve that is nonmonotonous on the x-axis. However, calculating the sum of two curves, where both curves are nonmonotonous on the x-axis, is not possible.

More Than One Curve in a Formula

If a formula includes more than one curve, Inspect interpolates all curves to a common x-axis dataset. This is demonstrated in the following example.

The data points of curves A, B, and C are marked with circles, squares, and diamonds, respectively. The points in the resulting curve are marked with plus signs. The formula used is $\langle A \rangle + \langle B \rangle + \langle C \rangle$. The resulting curve includes points of all three input curves:



Chapter 5: Curve Interpolation and Operations

Curve Operations

The Inspect tool creates an array of all x-coordinates of all curves that are used in a formula and interpolates those curves to obtain y-values for each of the new x-values added to each curve.

Handling Datasets in Formula Processing

The following examples illustrate the dataset handling method that the Inspect tool uses to work with more than one curve in a formula (see [Figure 8](#)).

Curve 1: $y = x^2$, linear scale on x

X	1	10	20	30	40	50	60	70
Y	1	100	400	900	1600	2500	3600	4900

Curve 2: $y = x^2$, logarithmic scale on x

X	1	2	4	8	16	32	64
Y	2	4	16	64	256	1024	4096

The combined set of x-coordinates needed to produce the resulting curve (the sum of both curves) is:

Resulting curve: $y = 2x^2$

X	1	2	4	8	10	16	20	30	32	40	50	60	64
---	---	---	---	---	----	----	----	----	----	----	----	----	----

The last point of Curve 1 ($x = 70$) is not included because no data is available beyond $x = 64$ in Curve 2.

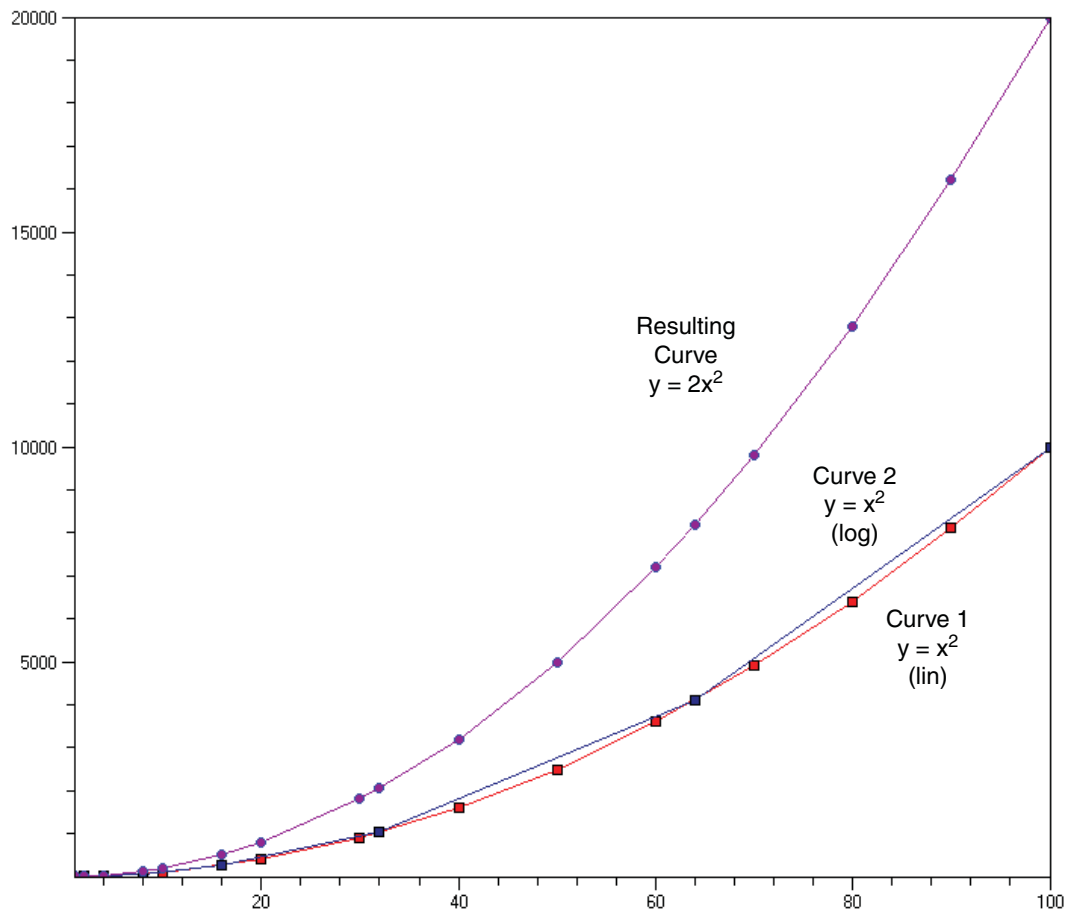
For y-values, interpolation is performed on Curve 1 and Curve 2 to fill the gaps and sum both curves. Therefore, y-values for this resulting curve are:

Y	2	8	32	128	200	512	800	1800	2048	3200	5000	7200	8192
---	---	---	----	-----	-----	-----	-----	------	------	------	------	------	------

Chapter 5: Curve Interpolation and Operations

Dataset Created for Result Curves

Figure 8 Resulting plot showing curves



Dataset Created for Result Curves

For each curve created by a formula that involves more than one curve, Inspect generates a new dataset. During the Inspect session, this dataset is stored in a special project called `AuxProject`. When the current project is saved in a file, the `AuxProject` is also saved; otherwise, this project with all its datasets is lost when Inspect is closed.

The datasets in `AuxProject` are handled in the same way as datasets from loaded data files.

Curve Handling on Interpolation

The Inspect tool handles both linear-scaled and logarithmic-scaled curves. Each curve is treated independently. Therefore, when working with two curves, one with linear scale and the other with logarithmic scale, the Inspect tool:

- Creates new points for the first curve for all x-values of the second curve using a linear interpolation method
- Creates new points for the second curve for all x-values of the first curve using a logarithmic interpolation method
- Operates with the common set of points

Determining the Scale (Linear or Logarithmic) of Curves

Deciding how to handle a curve involves analyzing its slope, which is defined as:

$$\frac{dy}{dx} = \frac{y_{i1} - y_i}{x_{i1} - x_i} \quad (1)$$

First, the curve is treated as linear, and the minimum (MinSlope) and maximum (MaxSlope) slopes are calculated.

Second, a quotient is created:

$$\frac{\text{MaxSlope}}{\text{MinSlope}} \quad (2)$$

The same calculation is performed by treating:

- The x-axis as logarithmically scaled
- The y-axis as logarithmically scaled
- Both axes as logarithmically scaled

Of these four values, the one closest to 1.0 indicates the best way of handling the curve.

6

Formulas and Macros

This chapter describes how to use formulas and macros in Inspect.

Using Formulas

The Inspect tool recognizes two variable types: curve and scalar. Mixed curve–curve and curve–scalar operations are evaluated as follows:

1. The range of the result of a curve–curve operation is the intersection of the x-range of the operands.
2. When one operand is a curve and the other is a scalar, the respective operation is performed as a scalar operation on each element of the curve operand.

The binary operators that can be used are +, −, *, /, and ^ (power operator).

Formula Library

The formula library allows you to perform some basic calculations on one or more curves. The result can be a new curve or a scalar value (see [Table 3 on page 48](#)). The following examples show how the formula library is used:

<code>sin (<curve_1> + 10)</code>	The result is a new curve. Inspect adds 10 to the y-value of each curve point from <code>curve_1</code> and computes the sinus.
<code>maxslope(<curve_1>)</code>	The result is a scalar value, which is the maximum slope of <code>curve_1</code> .

[Table 3](#) lists functions that create a new curve by applying a mathematical transformation to each element of the curve. These functions can also be applied to scalar values. [Table 4](#) lists special functions that either require more than one parameter or do not return a curve. [Table 5](#) lists functions that manage or compute fast Fourier transformation (FFT) and related operations.

Note:

A curve can be defined by one point only, in which case, the curve is treated as a scalar input. Some curves require as input a *curve only*, that is, a curve that has at least two points.

Table 3 Standard mathematical functions

Function	Input type	Output type	Description
acos	curve	curve	Returns the arc cosine. The returned angle [radian] is given in the range 0 (zero) to π .
acosh	curve	curve	Returns the inverse hyperbolic cosine. Curve values must be greater than or equal to 1.
asin	curve	curve	Returns the arc sine. The returned angle [radian] is given in the range from $-\pi/2$ to $\pi/2$.
asinh	curve	curve	Returns the inverse hyperbolic sine.
atan	curve	curve	Returns the arc tangent. The returned angle [radian] is given in the range from $-\pi/2$ to $\pi/2$.
atanh	curve	curve	Returns the inverse hyperbolic tangent. Curve values must be between -1 and 1 (excluding -1 and 1).
cbrt	curve	curve	Returns the cube root.
ceil	curve	curve	Rounds up each element to the smallest integer not less than itself.
cos	curve	curve	Returns the cosine.
cosh	curve	curve	Returns the hyperbolic cosine.
diff	curve only	curve	Returns the first derivative of the curve.
erf	curve	curve	Returns an error function of the curve values.
erfc	curve	curve	Returns the complementary error function of the curve values.
exp	curve	curve	Returns the number raised to the power of each curve value.
fabs	curve	curve	Returns the absolute value.

Chapter 6: Formulas and Macros

Using Formulas

Table 3 Standard mathematical functions (Continued)

Function	Input type	Output type	Description
floor	curve	curve	Rounds down each element to the largest integer not greater than itself.
gamma	curve	curve	Returns the Gamma function.
integr	curve only	curve	Returns the integral of the curve.
j0	curve	curve	Returns the Bessel function of the first kind of order 0.
j1	curve	curve	Returns the Bessel function of the first kind of order 1.
lgamma	curve	curve	Returns the natural logarithm of the absolute value of the Gamma function.
log	curve	curve	Returns the natural logarithm of the given curve.
log10	curve	curve	Returns the base 10 logarithm of the given curve.
sin	curve	curve	Returns the sine.
sinh	curve	curve	Returns the hyperbolic sine.
sqrt	curve	curve	Returns the square root.
tan	curve	curve	Returns the tangent.
tanh	curve	curve	Returns the hyperbolic tangent.
y0	curve	curve	Returns the Bessel function of the second kind of order 0.
y1	curve	curve	Returns the Bessel function of the second kind of order 1.

Table 4 Special functions

Function	Input type	Output type	Description
pow	curve, scalar	curve	Returns the curve raised to the power of the given scalar.
tangent	curve, scalar	curve	Returns a curve that is tangent to the given curve, at the given x-value.

Table 4 *Special functions (Continued)*

Function	Input type	Output type	Description
vecmax	curve	scalar	Maximum y-value.
vecmin	curve	scalar	Minimum y-value.
vecvalx	curve, scalar	scalar	The x-value at a given y.
vecvaly	curve, scalar	scalar	The y-value at a given x.
veczero	curve	scalar	The x-value at y = 0.

Table 5 *Fast Fourier transformation (FFT) and related functions*

Function	Input type	Output type	Description
cfftim	curve_real, curve_imaginary	curve	Returns the imaginary part of the FFT of the given complex curve.
cfftre	curve_real, curve_imaginary	curve	Returns the real part of the FFT of the given complex curve.
cifftim	curve_real, curve_imaginary	curve	Returns the imaginary part of the inverse FFT of the given complex curve.
cifftre	curve_real, curve_imaginary	curve	Returns the real part of the inverse FFT of the given complex curve.
fftabs	curve_real, curve_imaginary	curve	Returns a vector holding the absolute value of the given complex curve.
fftim	curve	curve	Returns the imaginary part of the FFT of the given curve.
fftre	curve	curve	Returns the real part of the FFT of the given curve.
ifftim	curve	curve	Returns the imaginary part of the inverse FFT of the given curve.
ifftre	curve	curve	Returns the real part of the inverse FFT of the given curve.

Using Macros

Macros can define complex formulas. Inspect expands a macro by using the actual arguments specified in the call to the macro (see [Figure 3 on page 33](#)).

In a macro definition, the argument type must be specified. Types can be curve or scalar. This information is needed to expand the macro into the correct formula.

The syntax for argument placeholder specification is `<c n>` for curves and `<s n>` for scalars, where `n` is an integer value used to distinguish between different arguments; `n` must start with 1.

In the Inspect macro parser, the macro prototype is not specified explicitly. It is determined automatically from the formula that defines the macro. The order of arguments is determined by their first appearance in the formula and not by numbers in the argument placeholders.

Example: ADD Macro

The macro `ADD` is defined as:

```
<c 1> + <c 2>
```

This macro adds two curves. The macro prototype looks like:

```
ADD(<CURVE>, <CURVE>)
```

The argument placeholder `<CURVE>` must be replaced by an actual curve name.

Example: DIFFMULT Macro

The macro `DIFFMULT` is defined as:

```
diff(<c 1>) + (<s 3> * <c 2>)
```

This macro takes the derivative of a curve `<c 1>` and adds to it a curve `<c 2>` multiplied by a scalar `<s 3>`. A call to this macro has the form:

```
DIFFMULT(<CURVE>, S, <CURVE>)
```

The argument placeholder `<CURVE>` must be replaced by an actual curve name, and `S` must be replaced by an expression that generates a scalar value.

7

Using the Scripting Language

This chapter describes the operations available using the Inspect scripting language.

Overview of the Scripting Language

In addition to the user interface, you can control the Inspect tool using a scripting language (see [Working With Scripts on page 22](#)). The scripting language allows you to manipulate and display data without using the user interface, and it is very useful for running complex calculations on datasets and displaying results. For example:

- Repeated manual actions can be recorded and run later by simple script invocation.
- Several computations using the formula library can be performed in one run.
- Results can be written automatically to a file.

You can write a script manually or create a script by recording actions performed in the user interface (see [Creating Scripts on page 22](#)).

The Inspect tool uses the tool command language (Tcl) as its scripting language. For more information about Tcl, go to <http://www.tcl.tk>.

Some commands have been added to Tcl (in the form of Tcl procedures) to perform application-specific actions. For more specific needs, you can create your own commands.

Most of the additional commands in Inspect return a status string. A return status not equal to 1 indicates an error. If an error occurs, Inspect prints an error message to the standard error output and terminates the execution of the script.

Note:

Arguments in braces are *optional*. The first term in the braces is the name of the argument, and the second term is the default value of the argument. For example, a command that has been defined as `command {arg def_value}` can be called as `command` (which is equivalent to `command def_value`) and also as `command other_value`.

General-Purpose Commands

This section describes general-purpose commands.

ft_scalar

```
ft_scalar variableName variableValue
```

Action Produces the following output line: DOE: *variableName variableValue*
If the current Inspect command file belongs to a Sentaurus Workbench project, then this output line results in the creation of a new Sentaurus Workbench extracted variable with the name *variableName* and the value *variableValue* (see *Sentaurus™ Workbench User Guide*, Extracted Variables).

Input *variableName*, name of the Sentaurus Workbench variable to extract
variableValue, value of this Sentaurus Workbench variable

Returns None

Reading and Writing Files

This section describes commands for reading and writing files.

cv_write

```
cv_write type fileName curveList
```

Action Writes (exports) the data of the specified curves to a file in the specified format.

Input *type*, output format to use: plt, xgraph, or xmgr
fileName, file to write
curveList, list of curve names

Returns Status of the write operation

fi_writeBitmap

```
fi_writeBitmap fileName
```

Action Writes the plot area to a PNG file.

Input *fileName*, file to write

Returns Status of the write operation

fi_writeEps

```
fi_writeEps fileName {orientation portrait} {height ""} {width ""}
```

Action Writes the plot area to an EPS file. This command is not generated automatically when script recording is switched on. If height or width is not specified, the actual plot size is taken into account. Some examples are:

```
fi_writeEps test.eps  
fi_writeEps test.eps landscape  
fi_writeEps test.eps landscape 200 100
```

Input *fileName*, file to write
 orientation, image orientation: portrait (default) or landscape
 height, height of the saved image in pixels
 width, width of the saved image in pixels

Returns Status of the write operation

fi_writePs

```
fi_writePs fileName {orientation portrait} {printSize US_LETTER} {height ""}  
{width ""} {offsetHeight ""} {offsetWidth ""} {sizeUnit ""}
```

Action Writes the plot area to a PostScript® file. This command is not generated automatically when script recording is switched on. When height or width is not specified, the actual plot size is taken into account. Some examples are:

```
fi_writePs test.ps  
fi_writePs test.ps landscape  
fi_writePs test.ps landscape DIN_A4  
fi_writePs test.ps portrait US_LETTER 450 300 5 5 m
```

Input *fileName*, file to write
orientation, image orientation: portrait (default) or landscape
printSize, page size: US_LETTER (default), DIN_A3, or DIN_A4
height, height of the saved image in size units (*sizeUnit*)
width, width of the saved image in size units (*sizeUnit*)
offsetHeight, vertical page offset in size units (*sizeUnit*)
offsetWidth, horizontal page offset in size units (*sizeUnit*)
sizeUnit, unit for *height*, *width*, *offsetHeight*, and *offsetWidth*:
i: inch (default), m: millimeter, c: centimeter

Returns Status of the write operation

graph_load

```
graph_load fileName
```

Action Loads the specified save file into Inspect. All currently loaded projects are deleted.

Input *fileName*, name of file to load

Returns Status of the load operation

graph_write

`graph_write fileName`

Action Saves the current state to a specified file.

Input *fileName*, name of file

Returns Status of the write operation

param_load

`param_load fileName`

Action Loads a parameter file.

Input *fileName*, name of file to load

Returns Status of the load operation

param_write

`param_write fileName`

Action Saves a parameter file.

Input *fileName*, name of file

Returns Status of the write operation

proj_getDataSet

`proj_getDataSet projectName dataSetId`

Action If no dataset is found, the return value is an empty list. For example, the following commands set the variable `x_data` to the values of the dataset `time` and the variable `y_data` to the values of the dataset `data_1` of node_A:

```
set x_data [proj_getDataSet "tutorial_ins" "time"]
set y_data [proj_getDataSet "tutorial_ins" "node_A data_1"]
```

Input *projectName*, name of project
dataSetId, name of a dataset including its group name if applicable

Returns List of all the values of the dataset

proj_getList

`proj_getList`

Action Returns a list of all projects. If no projects are found, an empty list is returned.

Input None

Returns List of all loaded projects

proj_getNodeList

`proj_getNodeList projectName`

Action Returns a list of group names of the given project. If no groups have been found, an empty list is returned.

Input *projectName*, name of project

Returns List of group names

proj_load

`proj_load fileName`

Action Loads a data file and creates a new project. The base name of the file is used as the project name (see [Data Formats on page 16](#)).

Input *fileName*, name of file

Returns Status of the load operation

proj_unload

`proj_unload projectName`

Action Deletes a project and all the project-related curves.

Input *projectName*, name of project

Returns Status of the delete operation

proj_write

`proj_write projectName fileName`

Action Writes a project to a specified file.

Input *projectName*, name of project
fileName, name of file

Returns Status of the write operation

Creating, Displaying, and Deleting Curves

A dataset used for curve creation is identified by its data path, which consists of the project name, the group name when the dataset belongs to a group, and the dataset name.

cv_create

```
cv_create curveName xDataPath yDataPath {axis y}
```

Action Creates a curve with the given name using the specified datasets without displaying it. The datasets must be already loaded; otherwise, an error is returned. For example, the following command creates a curve `mycurve` using the dataset `time` on the x-axis and the dataset `OuterVoltage` of the group `Gate` on the y-axis, with both datasets belonging to the project `nmos_n7_des`:

```
cv_create mycurve "nmos_n7_des time" "nmos_n7_des Gate OuterVoltage"
```

Input *curveName*, unique name for the new curve
xDataPath, x-dataset data path
yDataPath, y-dataset data path
axis, axis to use; the default is *y*

Returns Status of the create operation

cv_createDS

```
cv_createDS curveName xDataPath yDataPath {axis y}
```

Same as `cv_create` except that the curve is displayed. See [cv_create](#).

cv_createFromScript

```
cv_createFromScript curveName xdata ydata {axis y}
```

Action Creates a curve using the given name and data. If the number of values for x and y are not the same, the number of curve points is according to that of the smaller dataset. Curves created with this command are stored in `AuxProject`.
For example, the following command creates the curve `mycurve` defined by the specified data:

```
cv_createFromScript mycurve "0 1 2 3 4 5 6 7 8 9" "1 2 1 2 1 2 1 2 1 2"
```

Input *curveName*, unique name for the new curve
xdata, list of data to use for the x-dataset
ydata, list of data to use for the y-dataset
axis, axis to use: *y* (default) or *y2*

Returns Status of the create operation

cv_createWithFormula

```
cv_createWithFormula curveName formula xmin xmax ymin ymax
```

Action Computes a new curve using the formula applied to the data of the argument curves within the given range. Setting the range to any nonnumeric value (for example, `A`) instructs the Inspect tool to set no limit in the corresponding direction.
For example, the following command creates the curve `f3` using the entire data range of curves `f1` and `f2`:

```
cv_createWithFormula f3 "<f1>+<f2>+10" A A A A
```

Input *curveName*, unique name for the new curve
formula, formula or macro
xmin, *xmax*, *ymin*, *ymax*, range for which the formula is applied

Returns Status of the create operation

cv_delete

`cv_delete curveName`

Action Deletes a curve.

Input *curveName*, name of curve

Returns Status of the delete operation

cv_display

`cv_display curveName {axis y}`

Action Displays a curve using the specified y-axis.

Input *curveName*, name of curve to display
axis, axis to use; the default is *y*

Returns None

cv_logScale, cv_log10Scale

`cv_logScale curveName newCurveName {axis x}`
`cv_log10Scale curveName newCurveName {axis x}`

Action Creates a new curve where all values on a given axis are transformed to a log (log10) scale.

Input *curveName*, curve to transform
newCurveName, name of the new curve
axis, axis on which the curve is scaled; the default is *x*

Returns Status of the create operation

cv_split

`cv_split curveName axis newCurveList`

Action Splits the input curve into several curves at the points where the x-values are nonmonotonic, that is, $x[i + 1] < x[i]$. The number of names for the new curves must match the actual number of created curves; otherwise, an error is returned.
This command is similar to choosing **Curve > Transform > Suppress Backtrace** (see [Table 10 on page 126](#)). The difference is that this command creates a set of new curves. With **Suppress Backtrace** selected, the backtrace lines are suppressed only on the plot.

Input *curveName*, name of curve to split
axis, y-axis to map the new curves onto
newCurveList, list of names for new curves

Returns Status of the operation

cv_split_disc

`cv_split_disc curveName axis newCurveList`

Action Splits the input curve into several curves at the points where there are discontinuities, that is, $x[i + 1] \neq x[i]$ and $y[i + 1] \neq y[i]$.
The number of names for the new curves must match the actual number of created curves; otherwise, an error is returned.

Input *curveName*, name of curve to split
axis, y-axis to map the new curves onto
newCurveList, list of names for new curves

Returns Status of the operation

Changing Attributes

These commands change the attributes of the title, axes, curves, and legend.

cv_lineColor

```
cv_lineColor curveName color
```

Action Sets the color of the curve line.

Input *curveName*, name of curve
color, color of the curve line

Returns None

cv_lineStyle

```
cv_lineStyle curveName style
```

Action Sets the drawing style of the curve line.

Input *curveName*, name of curve
style, drawing style of the curve line: dashed, dotted, "long dashed", "long dotted", or solid

Returns None

cv_renameCurve

```
cv_renameCurve curveName newName
```

Action Renames a curve.

Input *curveName*, name of curve
newName, new name of curve

Returns None

cv_set_interpol

cv_set_interpol curveId axis type

Action Sets the interpolation method to be applied to each particular dataset of a curve.

Input *curveId*, curve identification
 axis, axis on which the interpolation is set: X or Y
 type, interpolation method to set: AUTO, LIN, or LOG

Returns None

cv_setCurveAttr

cv_setCurveAttr curveName legend color style width shape size outColor outWidth fillColor

Action Sets curve-drawing attributes.

Input *curveName*, name of curve
 legend, curve legend
 color, color of the curve line
 style, drawing style of the curve line: dashed, dotted, "long dashed", "long dotted", or solid
 width, width of the curve line
 shape, marker shape: none, circle, cross, diamond, plus, scross, splus, square, or triangle
 size, marker size
 outColor, color of the marker outline
 outWidth, width of the marker outline
 fillColor, fill color of the marker

Returns None

gb_setpreferences

`gb_setpreferences type val`

Action Sets new values for the preference options. The following options can be modified:
 precision: Defines the precision used to display coordinate values in the status line; any integer can be set.
 old_interpolation: Specifies whether the old interpolation is used to compute curves:
 1: Activates old interpolation.
 0: Deactivates old interpolation.

Input *type*, preference option to be modified
 val, new value for option

Returns None

gr_createLabel

`gr_createLabel label coordX coordY fontStr color`

Action Creates a label in the plot area.

Input *label*, label text
 coordX, x-coordinate
 coordY, y-coordinate
 fontStr, label font
 color, label color

Returns Label ID

gr_deleteLabel

`gr_deleteLabel labelId`

Action Deletes a label from the plot area.

Input *labelId*, label ID

Returns None

gr_formatAxis

`gr_formatAxis axis format`

Action Changes the format of the displayed axis.

Input *axis*, axis to be formatted
format, options are default, engineering, fixed, or scientific

Returns None

gr_mappedAxis

`gr_mappedAxis axis yesno`

Action Changes the visibility of an axis.

Input *axis*, specifies a y-axis: *y* or *y2*
yesno, specifies the axis visibility: True or False

Returns None

gr_precision

`gr_precision axis prec`

Action Changes the precision of a given axis.

Input *axis*, axis to be formatted
prec, numeric precision to be set for the axis

Returns None

gr_setAxisAttr

```
gr_setAxisAttr axis title tfont min max color width font angle div scale {tcolor}
```

Action Sets the axis attributes.

Input

- axis*, specifies an axis: X, Y, or Y2
- title*, axis title
- tfont*, font size of the axis title
- min*, *max*, minimum and maximum values of the axis
- color*, color of the axis
- width*, width of the axis line
- font*, font size of the tick label
- angle*, angle at which the tick labels are drawn
- div*, number of secondary ticks between the main ticks
- scale*, specifies linear (lin) or logarithmic (log) display of the axis
- tcolor*, color of the axis title

Returns None

gr_setGeneralAttr

```
gr_setGeneralAttr {showFrame true} {axesTight true} {backColor white}  
{selectColor cyan}
```

Action Sets the general attributes of the plot.

Input

- showFrame*, Boolean indicator of plot frame appearance; default is `true`
- axesTight*, Boolean indicator of the tightness of axes; default is `true`
- backColor*, plot background color; default is `white`
- selectColor*, color of the selected curve; default is `cyan`

Returns None

gr_setGridAttr

```
gr_setGridAttr {showGrid false} {gridAlign left} {minorTicks false}  
{gridStyle "short dashed"} {gridColor black} {gridWidth 1}
```

Action Sets the grid attributes of the plot.

Input *showGrid*, Boolean indicator of grid appearance; default is `false`
gridAlign, grid alignment: `left` (default) or `right`
minorTicks, Boolean indicator of the appearance of minor ticks; default is `false`
gridStyle, attribute of the grid style: `dashed`, `"dot-dashed"`, `dotted`,
`"long dashed"`, `"long dotted"`, `"short dashed"` (default), or `solid`
gridColor, color of the grid lines; default is `black`
gridWidth, thickness of the grid lines; default is `1`

Returns `None`

gr_setLegendAttr

```
gr_setLegendAttr {showFlag true} {fontName helvetica} {fontSize 10} {fontStyle {}}  
{backColor white} {foreColor black} {frameColor black} {frameWidth 1}  
{framePos right} {frameAnchor n}
```

Action Sets the attributes of the legend.

Input *showFlag*, Boolean indicator of legend appearance
fontSize, legend font size; default is `10`
fontStyle, legend font style: `bold`, `italic`, `overstrike`, or `underline`; default is an
empty list `{}`
backColor, legend background color; default is `white`
foreColor, legend foreground color; default is `black`
frameColor, legend frame color; default is `black`
frameWidth, legend frame width; default is `1`
framePos, legend frame position: `left`, `right` (default), `top`, `bottom`, `free`, or `plot`
frameAnchor, legend frame anchor: `n` (default), `e`, `s`, `w`, `ne`, `se`, `sw`, or `nw`

Returns `None`

gr_setLegendPos

```
gr_setLegendPos x y
```

Action Changes the position of the displayed legend in the plot area.

Input *x*, new x-coordinate for the legend in the plot area
 y, new y-coordinate for the legend in the plot area

Returns None

gr_setTitleAttr

```
gr_setTitleAttr title {fontSize 14} {just center}
```

Action Sets the title attributes.

Input *title*, title text
 fontSize, size of title font; default is 14
 just, title justification: center (default), left, or right

Returns None

Accessing Curve Data

These commands relate to accessing curve data.

cv_getVals

```
cv_getVals curveName
```

Action Returns a list of the x- and y-data. The x-data and y-data can be accessed using:

```
set xy [cv_getVals "f1"]  
set x [lindex $xy 0]  
set y [lindex $xy 1]
```

After this, the variables *x* and *y* hold the x- and y-datasets, respectively.

Input *curveName*, name of curve

Chapter 7: Using the Scripting Language

Accessing Curve Data

```
cv_getVals curveName
```

Returns List of the x- and y-data

cv_getValsX

```
cv_getValsX curveName
```

Action Returns a list that holds the x-dataset.

Input *curveName*, name of curve

Returns List of the x-data

cv_getValsY

```
cv_getValsY curveName
```

Action Returns a list that holds the y-dataset.

Input *curveName*, name of curve

Returns List of the y-data

cv_getXaxis

```
cv_getXaxis curveName
```

Action Returns the project name and the dataset ID using:

```
set answer [cv_getXaxis "myCurve"]  
set projectName [lindex $answer 0]  
set dataSetId [lindex $answer 1]
```

Input *curveName*, name of curve

Returns List with the project name and the dataset ID of the x-dataset

cv_getYaxis

`cv_getYaxis curveName`

Action Returns the project name and the dataset ID as for `cv_getXaxis`.
See [cv_getXaxis](#).

Input *curveName*, name of curve

Returns List with the project name and the dataset ID of the y-dataset

cv_printVals

`cv_printVals curveName`

Action Writes the x- and y-data of a curve to standard output.

Input *curveName*, name of curve

Returns List of the printed values

Transforming Curve Data

These commands change the way in which curve data is displayed without changing the curve datasets.

cv_abs

`cv_abs curveName axis`

Action Replaces negative values of the x- or y-dataset by their absolute values, depending on the axis argument. This command has the same effect as choosing **Curve > Transform > Abs X** or **Abs Y** (see [Table 10 on page 126](#)).

Input *curveName*, name of curve
axis, axis specifier

Returns Status of the operation

cv_delPts

`cv_delPts curveName indexList`

Action Deletes the points in the *indexList* from the set of points being displayed.

Input *curveName*, name of curve
indexList, list of indices of curve points

Returns Status of the delete operation

cv_inv

`cv_inv curveName axis`

Action Reflects a curve about the specified axis. This command is equivalent to choosing **Curve > Transform > Reflect X** or **Reflect Y** (see [Table 10 on page 126](#)).

Input *curveName*, name of curve
axis, axis specifier

Returns Status of the operation

cv_reset

`cv_reset curveName`

Action Restores the original appearance of the curve after a transformation. This command is equivalent to choosing **Curve > Restore Data** (see [Table 10 on page 126](#)).

Input *curveName*, name of curve

Returns Status of the operation

Extracting Parameters

These commands extract standard parameters of semiconductor devices. Some arguments of the commands have default values that are used when an argument is not specified.

f_Gamma

f_Gamma VT1 VT2 VB1 VB2 const

Action Computes the body-effect parameter at two different source–substrate voltages. The formula used to compute the body-effect parameter is:

$$\text{Gamma} = (VT2 - VT1) / ((\text{const} + VB2)^{1/2} - (\text{const} + VB1)^{1/2})$$

Input *VT1, VT2*, two threshold voltages
VB1, VB2, two different source–substrate voltages
const, $2\phi_F$ where ϕ_F is the Fermi-level potential; default value is 0.8 V

Returns Gamma [V^{1/2}] as a scalar or *f_error* in the case of an error

f_gm

f_gm curveName xmin xmax ymin ymax

Action Computes the maximum of transconductance for a given I_d – V_g curve.

Input *curveName*, curve used to calculate gm
xmin, xmax, ymin, ymax, range for computing the result; the default values correspond to the full curve range

Returns Value of gm [A/V] of the curve or *f_error* in the case of an error

f_hideInternalCurves

`f_hideInternalCurves`

Action Hides the internally used curves created by the commands of this section.
See [f_showInternalCurves](#) on page 75.

Input None

Returns None

f_IDSS

`f_IDSS curveName xmin xmax ymin ymax`

Action Computes the saturation current.

Input *curveName*, I_D - V_D curve at the fixed gate-source voltage
xmin, *xmax*, *ymin*, *ymax*, range for computing the result; the default values correspond to the full curve range

Returns Saturation current value or `f_error` in the case of an error

f_KP

`f_KP gm VDS`

Action Computes the transconductance parameter.

Input *gm*, transconductance value
VDS, drain source voltage; default is 0.1

Returns K_P $[A/V^2]$ value or `f_error` in the case of an error

f_Ron

`f_Ron curveName xmin xmax ymin ymax`

Action Computes the on-state resistance in the linear region.

Input *curveName*, I_d – V_d curve at the fixed gate–source voltage
xmin, *xmax*, *ymin*, *ymax*, range for computing the result; the default values correspond to the full curve range

Returns Value of R_{on} [k Ω] or `f_error` in the case of an error

f_Rout

`f_Rout curveName xmin xmax ymin ymax`

Action Computes the output resistance in the saturation region.

Input *curveName*, I_d – V_d curve at the fixed gate–source voltage
xmin, *xmax*, *ymin*, *ymax*, range for computing the result; the default values correspond to the full curve range

Returns Value of R_{out} [k Ω] or `f_error` in the case of an error

f_showInternalCurves

`f_showInternalCurves axis`

Action Displays the internally used curves created by the commands of this section.
See [f_hideInternalCurves](#) on page 74.

Input *axis*, axis to use; default is `left`

Returns None

f_TetaG

f_TetaG VT gm idvgs vgsvgs xmin xmax ymin ymax

Action	Computes the mobility modulation TetaG using the formula: $TetaG = gm(VGS_{low}) / ID(VGS_{high}) - 1 / (VGS_{high} - VT)$
Input	<i>VT</i> , threshold voltage value <i>gm</i> , transconductance value <i>idvgs</i> , I_d - V_g curve <i>vgsvgs</i> , V_g - V_g curve <i>xmin</i> , <i>xmax</i> , <i>ymin</i> , <i>ymax</i> , range for computing the result; the default values correspond to the full curve range
Returns	Value of TetaG [V^{-1}] or <i>f_error</i> in the case of an error

f_VT

f_VT curveName xmin xmax ymin ymax

Action	Computes the threshold voltage [V] of the given curve. The formula used to compute the threshold voltage is: $VT = \text{intercept}(\text{maxslope}(\text{curve}))$ Example 1: This statement computes V_{th} using default values for the range: <code>set vt1 [f_VT idvgs]</code> Example 2: This statement computes V_{th} using <i>xmin</i> =0.1 <i>xmax</i> =0.3 and default values for the y-range: <code>set vt2 [f_VT idvgs 0.1 0.3]</code>
Input	<i>curveName</i> , name of curve <i>xmin</i> , <i>xmax</i> , <i>ymin</i> , <i>ymax</i> , range for computing the result; the default values correspond to the full curve range
Returns	Threshold voltage value or <i>f_error</i> in the case of an error

f_VT1

`f_VT1 curveName xmin xmax ymin ymax`

Action Computes the threshold voltage [V] of the given curve.
 V_{th} is typically extracted at $I_d = 0.1 \mu A/\mu m$.

Input *curveName*, name of curve
xmin, *xmax*, *ymin*, *ymax*, range for computing the result; the default values correspond to the full curve range

Returns Threshold voltage value or `f_error` in the case of an error

f_VT2

`f_VT2 curveName`

Action Computes the threshold voltage [V] of the given curve. The method used to extract V_{th} is the intersection of `MaxSlope` and `MinSlope` lines in the log of the given curve.

Input *curveName*, name of curve

Returns Threshold voltage as a scalar value or `f_error` in the case of an error

Computing

This section describes computing commands.

cv_compute

`cv_compute formula xmin xmax ymin ymax`

Action Computes a scalar value using the formula.

Input *formula*, string with the formula to evaluate
xmin, *xmax*, *ymin*, *ymax*, range for which the formula is applied

Returns Scalar computation result

cv_getZero

`cv_getZero curveName xmin xmax ymin ymax`

Action Computes the x-coordinate of the point where the curve intersects the x-axis. If the curve does not cross the x-axis, an empty string is returned.

Input *curveName*, name of curve
xmin, xmax, ymin, ymax, range for which the command applies

Returns The x-value where the curve intersects the x-axis

macro_define

`macro_define macroName macroDef`

Action Defines a macro, which can be used later for computations.

Input *macroName*, name of the macro
macroDef, macro definition

Returns Status of the operation

Controlling Scripts

This section describes commands for controlling scripts.

script_break

`script_break`

Action Suspends the execution of a script and passes control to the user interface. The script execution can be resumed by choosing **Script > Continue Script** (see [Table 11 on page 127](#)).

Input None

Returns None

script_exit

`script_exit`

Action Stops the execution of a script and exits Inspect.

Input None

Returns None

script_sleep

`script_sleep sec`

Action Stops the execution of a script for a given number of seconds.

Input *sec*, time in seconds

Returns None

Examples of Using the Scripting Language

This section presents examples of how to use the scripting language.

Computing the Dose of Implanted Arsenic

If `As_Implant` is the name of an As profile previously created, compute the dose of implanted As by integrating the profile. Limit the integration to portions of the curve with a concentration larger than $1e14$ but without other limitations in depth or maximum concentration value:

```
set Dose_As [cv_compute "vecmax(integr(<n30_sd_Arsenic_Implant>))"  
A A 1e14 A]
```

If `IdVg` is the name of an I_{ds} - V_{gs} curve previously created, compute a transconductance curve using `diff`. Limit the computation to the window in the I_d - V_g curve defined by $V_{min} = 1.0$ V, $V_{max} = 4.0$ V, $I_{d_min} = 1e-10$, and $I_{d_max} = 5e-6$:

```
set gm [cv_compute "vecmax(diff(<IdVg>))" 1.0 4.0 1e-10 5e-6]
```

Creating a Macro to Compute V_t

Create a macro to compute V_t from the maximum of the second derivative of an I_d - V_g curve. Use $\langle c \ n \rangle$ as placeholders for curves and $\langle s \ n \rangle$ for scalars, where n represents the argument used in the macro and must start at 1. In the example, $\langle c \ 1 \rangle$ should be an I_d - V_g curve and $\langle s \ 2 \rangle$ is a multiplication factor:

```
macro_define Vt2d "<s 2>*vecvalx(diff(diff(<c 1>)),  
    0.999*vecmax(diff(diff(<c 1>))))"
```

If I_{dVg} is the name of an I_{ds} - V_{gs} curve previously created, use the macro created to compute V_t in mV:

```
set Vt2 [cv_compute "Vt2d(<IdVg>,1e3)" A A A A]
```


8

Working With Script Libraries

This chapter describes how to work with script libraries in the Inspect tool.

The Inspect scripting language is complemented by libraries that provide additional functionality for specific operations such as curve comparison.

Loading Libraries

You use the `load_library` command to load libraries:

```
load_library libraryName
```

where `libraryName` is a library identifier.

This command makes available all the functionality provided by the specified library.

All commands of a particular library have a common prefix, for example, `iccap` for commands provided by the IC-CAP model parameter extraction library (see [IC-CAP Model Parameter Extraction Library on page 118](#)).

Adding a Site Library

The `$STROOT_LIB/inspectlib` directory stores all libraries as well as the `lib_index` file, which provides an index of all available libraries.

To add a library, the administrator (a person with write permissions to the TCAD distribution directory `$STROOT`) copies the library to the `$STROOT_LIB/inspectlib` directory and enters text in the index that describes the new library. The following fields must be provided:

```
<library_name> <library_prefix> <library_filename>
```

Chapter 8: Working With Script Libraries

Extraction Library

where:

- `<library_name>` is the name specified to call this library.
- `<library_prefix>` is the prefix used for all commands.
- `<library_filename>` is the name of the file where all commands are implemented.

Extraction Library

The commands provided by this library extract parameters from I–V curves. You can load the library with the command:

```
load_library EXTRACT
```

The library is located at `$STROOT/$STRELEASE/lib/inspectlib/EXTRACT.tcl`. If you need to customize the library, you can create a local copy of the library and edit the scripts. In this case, the local version is loaded by sourcing the script:

```
source EXTRACT.tcl
```

cv_linTransCurve

This command applies a linear transformation to the x- and y-values of a curve. It is called using:

```
cv_linTransCurve <Curve> <Xm> <Xb> <Ym> <Yb> <Axis>
```

where:

- `<Curve>` is the name of the curve.
- The x- and y-values of the curve are replaced by the transformed values given by $X' = X * X_m + X_b$ and $Y' = Y * Y_m + Y_b$, respectively.
- `<Axis>` can be either `y` or `y2`, and determines on which y-axis the transformed curve is displayed.

Example

Shift an I_d – V_{gs} curve by 0.55 V:

```
cv_linTransCurve IdVgs 1 0.55 1.0 0.0 y
```

cv_scaleCurve

This command scales the x- and y-values of a curve. It is called using:

```
cv_scaleCurve <Curve> <XFactor> <YFactor> <Axis>
```

where:

- <Curve> is the name of the curve.
- The x- and y-values of the curve are multiplied by <XFactor> and <YFactor>, respectively.
- <Axis> can be either *y* or *y2*, and determines on which y-axis the scaled curve is displayed.

Example

Scale an I_d - V_{gs} curve from A/ μ m to mA/mm:

```
cv_scaleCurve IdVgs 1 1e6 y
```

ExtractBVi

Breakdown curves sometimes exhibit a pronounced snapback, in which case, another relevant definition is the bias voltage at which the current reaches a certain level. This type of extraction is performed with the `ExtractBVi` command. It is called using:

```
ExtractBVi <Name> <Curve> <Ilevel>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the curve.
- <Ilevel> refers to the mentioned current level.

Example

```
ExtractBVi BVcboi VcIc 1e-12
```

results in output such as:

```
DOE: BVcboi 9.09e+00  
BVi: 9.09e+00
```

ExtractBVv

The breakdown voltage can be defined as the maximum voltage that can be applied to a contact. The `ExtractBVv` command extracts this value. It is called using:

```
ExtractBVv <Name> <Curve> <Sign>
```

where:

- `<Name>` is the name of the extracted parameter.
- `<Curve>` is the name of the curve.
- `<Sign>` can take the values `+1` (n-p-n) or `-1` (p-n-p), and distinguishes different types of bipolar transistor. (In general, specify `-1` if the breakdown occurs at a negative bias.)

Example

```
ExtractBVv BVcboV VcIc 1.0
```

results in output such as:

```
DOE: BVcboV 9.09e+00  
BVv: 9.09e+00 V
```

ExtractEarlyV

This command extracts the Early voltage from an I_C - V_{CE} curve. It is called using:

```
ExtractEarlyV <Name> <Curve> <Vtarget>
```

where:

- `<Name>` is the name of the extracted parameter.
- `<Curve>` is the name of the curve.
- `<Vtarget>` is the bias point at which the slope of the I_C - V_{CE} curve is determined for the computation of the Early voltage.

Example

```
ExtractEarlyV Va IcVc 1.25
```

results in output such as (where R_O is the output resistance and V_A is the Early voltage):

```
DOE: Ro 3.283e+04  
DOE: Va -1.836e+01
```

ExtractGm

This command extracts the maximum transconductance from an I_d - V_{gs} curve. It is called using:

```
ExtractGm <Name> <Curve> [<Type>]
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is to the name of the I_d - V_{gs} curve.
- See [ExtractVtgm on page 88](#) for details about Type.

Example

```
set gm [ExtractGm gmLin IdVg]
```

results in output such as:

```
DOE: gmLin 1.123e-04 gm: 1.123e-04  
S/um Max gm is at Vg= 0.540 V
```

ExtractGmb

This command is the same as `ExtractGm` except that the `ExtractGmb` command uses parabolic interpolation to find the gate bias at which the maximum transconductance occurs (see [ExtractGm](#)).

For I_d - V_{gs} curves with a limited number of gate-bias sample points, better accuracy is achieved with the `ExtractGmb` command.

ExtractIoff

This command extracts the drain leakage current from an I_d - V_{gs} curve. It is called using:

```
ExtractIoff <Name> <Curve> <Voff>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the I_d - V_{gs} curve (computed for a high drain bias).
- <Voff> defines the gate voltage at which the drain leakage current is extracted, typically, at a small nonzero value to avoid noise.

Chapter 8: Working With Script Libraries

Extraction Library

Example

```
if { $Type == "nMOS" } { set SIGN 1.0 } \  
else { set SIGN -1.0 } \  
set Ioff [ExtractIoff Ioff [expr $SIGN*1e-4]]
```

results in output such as:

```
DOE: Ioff 5.167e-11  
Ioff: 5.167e-11 A/um
```

ExtractMax

This command extracts the maximum of a curve. It is called using:

```
ExtractMax <Name> <Curve>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the curve.

Example

```
set IdSat [ExtractMax IdSat IdVg]
```

results in output such as:

```
DOE: IdSat 4.028e-04  
Max: 4.028e-04
```

ExtractRon

This command extracts the on-state resistance from an I_d – V_{ds} curve. It is called using:

```
ExtractRon <Name> <Curve> <Von>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the I_d – V_{ds} curve (computed for a high gate bias).
- <Von> defines the drain voltage at which the on-state resistance is extracted, typically, well beyond saturation.

Example

```
set Ron [ExtractRon Ron IdVd 1.1]
```

results in output such as:

```
DOE: Ron 14909.555  
Ron: 14909.555 Ohm um
```

ExtractSS

This command extracts the subthreshold voltage swing from an I_d - V_{gs} curve. It is called using:

```
ExtractSS <Name> <Curve> <Vgo>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the I_d - V_{gs} curve.
- <Vgo> defines the gate voltage at which the slope is extracted. It should be a value well below the threshold voltage.

Note:

The slope might be *noisy* at the beginning of the curve or at very low current levels, so better results are often obtained when setting $v_{go} > 0$ V.

Example

```
set SS [ExtractSS SSlin IdVg($N) 0.01]
```

results in output such as:

```
DOE: SSlin 79.758  
SS (subthreshold voltage swing): 79.758 mV/dec
```

ExtractValue

This command extracts the y-value at a given x-point. It is called using:

```
ExtractValue <Name> <Curve> <Xo>
```

where:

- <Name> is the name of the extracted parameter.
- <Curve> is the name of the curve.

Chapter 8: Working With Script Libraries

Extraction Library

- `<Xo>` defines the x-point at which the value is extracted.

Example

```
set CggP [ExtractValue CgP Cgg 1.2]
```

results in output such as:

```
DOE: CgP 1.426e-15
CgP: 1.426e-15
```

Here, `Cgg` is the name of the Inspect total gate–capacitance versus the gate–voltage curve.

ExtractVtgm

This command extracts the threshold voltage from an I_d – V_{gs} curve using the maximum transconductance method. It is called using:

```
ExtractVtgm <Name> <Curve> [<Type>]
```

where:

- `<Name>` is the name of the extracted parameter as it appears in the Variable Values column of Sentaurus Workbench.
- `<Curve>` is the name of the I_d – V_{gs} curve.
- `<Type>` specifies the transistor type, which can be one of the following values:
 - `nMOS` or `nMOSneg` for NMOSFETs with a positive or negative drain current convention.
 - `pMOS` or `pMOSneg` for PMOSFETs with a positive or negative drain current convention.

The MOSFET threshold and transconductance extraction commands require prior knowledge of the transistor type and the sign convention for the drain current.

If `<Type>` is omitted, then the transistor type is determined internally by analyzing the first and last points of the given curve.

The command `ExtractVtgm` passes the extracted value to Sentaurus Workbench and prints it to the log file. It also returns the value to Inspect.

Example

```
set Vt [ExtractVtgm Vtgm IdVg]
```

results in output such as:

```
DOE: Vtgm 0.392
Vt (Max gm method): 0.392 V
```

ExtractVtgmb

This is the same as `ExtractVtgm` except that the `ExtractVtgmb` command uses parabolic interpolation to find the gate bias at which the maximum transconductance occurs (see [ExtractVtgm on page 88](#)).

For I_d – V_{gs} curves with a limited number of gate–bias sample points, better accuracy is achieved with the `ExtractVtgmb` command.

ExtractVti

This command extracts the gate voltage from an I_d – V_{gs} curve at which the drain current exceeds a given current level. It is called using:

```
ExtractVti <Name> <Curve> <Ilevel>
```

where:

- `<Name>` is the name of the extracted parameter.
- `<Curve>` is the name of the I_d – V_{gs} curve.
- `<Ilevel>` defines the drain current level at which to extract the gate voltage.

Example

```
set Vti [ExtractVti Vti IdVg 1e-7]
```

results in output such as:

```
DOE: Vti 1.476  
Vti (Vg at Io=1.000e-06): 1.476 V
```

FilterTable

The `FilterTable` command processes data from the Sentaurus Workbench Family Tree to create a plot of one Sentaurus Workbench parameter as a function of another Sentaurus Workbench parameter for a certain subset of experiments. Threshold voltage roll-off plots are a typical application of this utility.

To better understand this utility, it is helpful to first consider the kind of data on which it is designed to operate.

In an Inspect script, you can use the dynamic preprocessing feature of Sentaurus Workbench `@<parameter_name>:all@` to access a list of input parameters and extracted values for all Sentaurus Workbench experiments.

Chapter 8: Working With Script Libraries

Extraction Library

For example:

```
set Types [list @Type:all@]
set Lgs   [list @lgate:all@]
set Vts   [list @Vt:all@]
set Ids   [list @Id:all@]
...
```

Here, the Tcl list `Types` contains, for all experiments, the values of the Sentaurus Workbench input parameter `Type`, which for example can take the value `nMOS` or `pMOS`, depending on whether an NMOS or a PMOS structure is created in this experiment.

Similarly, the Tcl list `Lgs` contains for all experiments a *parallel* list of values of another Sentaurus Workbench input parameter, which for example contains the value of the gate length of the given MOSFETs. The corresponding extracted parameter can be accessed in the same way. The Tcl lists `Vts` and `Ids` can contain the extracted values for the threshold voltage and the drain current for each respective experiment.

Note:

The values in the various lists might or might not be numeric, and they might not be ordered.

Syntax of FilterTable

The `FilterTable` command takes lists of Sentaurus Workbench parameters as arguments. The first two lists identify the x- and y-values, which will be processed to create a plot. The subsequent arguments control the conditions an experiment must fulfill to be included in the plot. These conditions are defined using optional pairs of a target value and a Sentaurus Workbench list.

The syntax of `FilterTable` is:

```
FilterTable XList YList [ConditionTarget1 ConditionList1] \
  [ConditionTarget2 ConditionList2] \
  [ConditionTarget3 ConditionList3] [...]
```

The command returns two lists of values:

- The first list contains a subset of the `XList`. The subset is restricted to the selected experiments. The values are given in ascending order.
- The second list contains the corresponding values of the `YList`.

In addition, `FilterTable` ignores all entries of `YList` that contain a nonnumeric value. You can use this feature to omit failed extractions.

Chapter 8: Working With Script Libraries

The extend Library

In the tool input file that performs the extraction (for example, a previous Inspect instance), use the `#set` directive to preset the extracted variable to the value `x`:

```
#set Vt x
...
set Vt [ExtractVtgmb Vt IdVg]
```

The actual extraction process, here using the `ExtractVtgmb` command, overwrites the preset value `x` with the actual value. However, if the extraction process fails, the preset value persists.

For example, after preprocessing, Sentaurus Workbench preprocessor references such as `@Type:all@` are expanded and the resulting preprocessed file can look like:

```
set Types [list nMOS nMOS nMOS nMOS pMOS pMOS pMOS pMOS]
set Lgs [list 0.090 0.045 0.130 0.065 0.065 0.045 0.130 0.090]
set Vts [list 0.424 0.313 0.414 0.408 -0.344 -0.232 x -0.374]

set XYLists [FilterTable $Lgs $Vts "nMOS" $Types]
cv_createFromScript Vt_vs_Lg_nMOS [lindex $XYLists 0] \
[lindex $XYLists 1] y
cv_display Vt_vs_Lg_nMOS y

set XYLists [FilterTable $Lgs $Vts "pMOS" $Types]
cv_createFromScript Vt_vs_Lg_pMOS [lindex $XYLists 0] \
[lindex $XYLists 1] y
cv_display Vt_vs_Lg_pMOS y
```

This script creates two separate V_t roll-off curves: one for all `nMOS` experiments and one for all `pMOS` experiments. The values are shown in order and the data point for `Type=pMOS` and `Lg=0.130`, for which the extraction failed (`Vt=x`), is omitted.

The extend Library

The `extend` library implements high-level commands to provide:

- Better control of curve attributes (`cv_autoIncrStyle`, `cv_disp`, `cv_nextColor`, `cv_nextSymbol`, `cv_setFillColor`)
- Curve manipulation (`cv_addCurve`, `cv_addDataset`, `cv_linTrans`, `cv_monotonicX`, `cv_scale`, `cv_sort`)
- Additional curve information (`cv_getGlobalExtrema`, `cv_getLocalExtrema`, `cv_getNames`, `cv_getRange`, `cv_getXmax`, `cv_integrate`, `cv_linFit`)
- Extraction of dataset information (`ds_getValue`, `proj_datasetExists`)
- A simple debug print function (`dbputs`)

Chapter 8: Working With Script Libraries

The extend Library

- Functions to work with lists (`ldiff`, `lintersect`, `ltranspose`, `lunion`)
- An ASCII file import filter (`fi_readTxtFileHeader`)

You can load the library with the command:

```
load_library extend
```

The library is located at `$STROOT/$STRELEASE/lib/inspectlib/extend.tcl`.

If you need to customize the library, you can create a local copy of the library and edit the scripts. In this case, the local version is loaded by sourcing the script:

```
source extend.tcl
```

The commands of the library are described in the following sections. If a command is applied to a curve, the creation of the curve is not mentioned explicitly in the examples for brevity. For test purposes, curves can be created easily with the following line:

```
cv_createFromScript c1 {0 1} {1 2}
```

Note:

Arguments in braces are *optional*. The first term in the braces is the name of the argument, and the second term is the default value of the argument. For example, a command that has been defined as `command {arg def_value}` can be called as `command` (which is equivalent to `command def_value`) and also as `command other_value`.

cv_addCurve

```
cv_addCurve cname cname2
```

Action	Adds the y-values of the curve <i>cname2</i> to the y-values of the curve <i>cname</i> .
---------------	--

Input	<i>cname</i> , <i>cname2</i> , name of curves
--------------	---

Returns	None
----------------	------

Example

```
cv_createFromScript c1 {0 1} {1 2}
cv_createFromScript c2 {0 1} {3 4}
cv_addCurve c1 c2
puts "y: [cv_getValsY c1]"
> y: 4 6
```

cv_addDataset

`cv_addDataset cname xdset ydset`

Action Adds the y-values of a dataset to an existing curve.

Input *cname*, name of curve to which datasets will be added
xdset, dataset name of the x-values to be added
ydset, dataset name of the y-values to be added

Returns None

Example

```
# sum the total currents of nContact and nContact2
cv_addDataset iv "n4_des pContact OuterVoltage" \
  "n4_des nContact TotalCurrent"
cv_addDataset iv "n4_des pContact OuterVoltage" \
  "n4_des nContact2 TotalCurrent"
```

cv_angularMap

`cv_angularMap cname {astart 0} {aend 360}`

Action Maps a periodic curve to a fixed angular range of *astart* to *aend*.
For angular data, you might want to reduce all data points to the first period. For example, if a full circle with 0..360° will be plotted, but datapoints with x-values higher than 360 exist, these should be mapped to the first period, that is, the y-value at x=361 will be added to the datapoint x=1.

Input *cname*, name of curve
astart, start of the angular range; default is 0
aend, end of the angular range; default is 360

Returns None

Example

```
cv_createFromScript a {0 1 2 90 91 92} {1 2 3 4 5 6}
cv_angularMap a 0 90
puts [cv_getVals a]
-> {0 1 2 90} {1 7 9 4}
```

cv_autoIncrStyle

```
cv_autoIncrStyle {stylelist {color fillColor line symbol}} | off
```

Action Sets the curve attributes to be incremented by one whenever a curve is displayed using `cv_disp`. The attributes are incremented in the order given by *stylelist*.

Input *stylelist*, list of options; default is {color fillColor line symbol}
off, switches off the automatic increment feature

Returns None

Example

```
# First increment color. If all colors are used, increment symbol
# and start with first color again.
cv_autoIncrStyle {color symbol}
cv_disp c1
cv_disp c2
```

cv_disp

```
cv_disp cname {label ""} {axis "y"}
```

Action Displays a curve using the specified label and axis. Curve attributes are incremented by default, such that each displayed curve can be easily distinguished.

Additional control of the curve attributes is given by the following commands:

`cv_autoIncrStyle`, `cv_nextColor`, `cv_nextLine`, `cv_nextSymbol`, `cv_resetColor`,
`cv_resetFillColor`, `cv_resetLine`, `cv_resetStyle`, `cv_resetSymbol`

Input *cname*, name of curve
label, specifies the curve label to be displayed in the legend; default is the curve name
axis, specifies the axis to use: *y* (default) or *y2*
Controlling attributes manually makes most sense when `cv_autoIncrStyle` is switched off.

Returns None

Example

```
cv_disp iv "simulated IV" y
```

cv_exists

`cv_exists cname`

Action	Checks whether a curve exists.
Input	<i>cname</i> , name of curve
Returns	1 (the curve exists) or 0 (the curve does not exist)

Example

```
if {[cv_exists iv]} {puts "curve iv exists"}
```

cv_getGlobalExtrema

`cv_getGlobalExtrema cname {type max}`

Action	Returns the global maximum or minimum of a curve as a list.
Input	<i>cname</i> , name of curve <i>type</i> , specifies either the global maximum (<i>max</i>) or the global minimum (<i>min</i>); default is <i>max</i>
Returns	If <i>type</i> equals <i>max</i> : {xmax ymax} If <i>type</i> equals <i>min</i> : {xmin ymin}

Example

```
set cmin [cv_getGlobalExtrema iv min]
puts "The minimum of the iv-curve is [lindex $cmin 1] and occurred at
    [lindex $cmin 0]"
```

cv_getLocalExtrema

`cv_getLocalExtrema cname {type max}`

Action Returns all local maxima or minima of a curve as a list.

Input *cname*, name of curve
type, specifies either the local maxima (`max`) or local minima (`min`); default is `max`

Returns If *type* equals `max`: `{{xmax1 ymax1} {xmax2 ymax2} ...}`
If *type* equals `min`: `{{xmin1 ymin1} {xmin2 ymin2} ...}`

Example

```
set cmax [cv_getLocalExtrema iv "max"]
puts "All maxima of the iv-curve: $cmax"
```

cv_getNames

`cv_getNames`

Action Returns all existing curve names.

Input None

Returns List of curve names

Example

```
puts "the following curves exist currently: [cv_getNames]"
```

cv_getRange

`cv_getRange cname`

Action Returns the x- and y-range of a curve as a list.

Input *cname*, name of curve

Returns {xmin,xmax,ymin,ymax}

Example

```
puts "{xmin,xmax,ymin,ymax}: [cv_getRange iv]"
```

cv_getXmax

`cv_getXmax cname`

Action Returns the upper boundary of the x-values of a curve.

Input *cname*, name of curve

Returns xmax

Example

```
puts "upper boundary of the x values of iv: [cv_getXmax iv]"
```

cv_getXmin

`cv_getXmin cname`

Action Returns the lower boundary of the x-values of a curve.

Input *cname*, name of curve

Returns xmin

Example

```
puts "lower boundary of the x values of iv: [cv_getXmin iv]"
```

cv_getYmax

`cv_getYmax cname`

Action Returns the upper boundary of the y-values of a curve.

Input *cname*, name of curve

Returns ymax

Example

```
puts "upper boundary of the y values of iv: [cv_getYmax iv]"
```

cv_getYmin

`cv_getYmin cname`

Action Returns the lower boundary of the y-values of a curve.

Input *cname*, name of curve

Returns ymin

Example

```
puts "lower boundary of the y values of iv: [cv_getYmin iv]"
```

cv_integrate

```
cv_integrate formula {xstart {}} {xend {}} {mode {}} {xdigits {}}
```

Action Performs integration of a formula, and returns the integration value.

Input *formula*, formula to be integrated as it is specified for `cv_createWithFormula`
xstart, start of the integration interval; if not specified, the start of the curve is used
xend, end of the integration interval; if not specified, the end of the curve is used
mode, defines the integration mode:

- If not specified, then it is set to {}, which performs the Inspect internal integration using `integr()`.
- `sumup` sums all y-values.
- `trapez` performs the integration using the trapezoidal rule.

xdigits, specifies the number of digits to determine whether two x-values are identical

When the formula contains more than one curve and these curves have different sets of x-values, summation is performed over all x-values.

Returns Integration value

Example

```
# P_t is the time-dependent power
puts "energy in the first second is: [cv_integrate P_t 0 1 trapez]"
```

cv_isVisible

```
cv_isVisible cname
```

Action Checks whether a curve is displayed.

Input *cname*, name of curve

Returns 1 if curve is currently displayed; otherwise, 0

Example

```
if {[cv_isVisible iv]} {puts "curve iv visible"}
```

cv_linFit

`cv_linFit formula {xstart {}} {xend {}}`

Action Performs a linear fit $y = A + B \cdot x$ to a curve formula.

Input *formula*, formula to be fitted, as it is specified for `cv_createWithFormula`
xstart, start of the fit interval
xend, end of the fit interval

Returns Fit results as a list:

- (Estimate of) Intercept A
- (Estimate of) Slope B
- Standard deviation of y relative to the fit correlation coefficient R^2
- Number of degrees of freedom df
- Standard error of intercept A
- Significance level of A
- Standard error of slope B
- Significance level of B

Example

```
set res [cv_linFit "log(<cname>)"]
```

cv_linTrans

`cv_linTrans cname xm {xb 0} {ym 1} {yb 0}`

Action Scales a curve linearly, and replaces x with $xm \cdot x + xb$ and y with $ym \cdot y + yb$.

Input *cname*, name of curve
xm, slope of the x-values
xb, offset of the x-values
ym, slope of the y-values
yb, offset of the y-values

Returns None

Chapter 8: Working With Script Libraries

The extend Library

Example

```
# Scaling an IV curve given in A over V, to mA over mV.
# In addition, the curve had a current offset of 2A, which you want
# to remove.
cv_linTrans iv 1e3 0 1e3 -2000
```

cv_monotonicX

`cv_monotonicX cname`

Action Extracts the part of a curve where the x-values increase monotonically to the maximal x-value.

Input *cname*, name of curve

Returns None

Example

```
cv_createFromScript iv {0 1 2 0 2 4} {1 2 3 4 5 6}
cv_monotonicX iv
puts "values: [cv_getVals iv]"
-->values: {0 2 4} {4 5 6}
```

cv_nextColor

`cv_nextColor {cindex ""}`

Action Sets the next color of the curve from the `extend::COLORPALETTE` list.

Input *cindex*, if *cindex* is specified, the specified entry from the `extend::COLORPALETTE` list is taken; otherwise, the next entry is chosen
Controlling attributes manually makes most sense when `cv_autoIncrStyle` is switched off.

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextColor
cv_disp iv2
```

cv_nextLine

```
cv_nextLine {cindex ""}
```

Action	Sets the next line style of the curve from the <code>extend::LINEPALETTE</code> list.
Input	<i>cindex</i> , if <i>cindex</i> is specified, then the specified entry from the <code>extend::LINEPALETTE</code> list is taken; otherwise, the next entry is chosen. If the last line style is reached, the first line style is returned again. Controlling attributes manually makes most sense when <code>cv_autoIncrStyle</code> is switched off.
Returns	None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextLine
cv_disp iv2
```

cv_nextSymbol

```
cv_nextSymbol {cindex ""}
```

Action	Sets the next symbol type of the curve from the <code>extend::SYMBOLPALETTE</code> list.
Input	<i>cindex</i> , if <i>cindex</i> is specified, then the specified entry from the <code>extend::SYMBOLPALETTE</code> list is taken; otherwise, the next entry is chosen. If the last symbol is reached, the first symbol is returned again. Controlling attributes manually makes most sense when <code>cv_autoIncrStyle</code> is switched off.
Returns	None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextSymbol
cv_disp iv2
```

cv_resetColor

cv_resetColor

Action Resets the color to the default entry that equals the first entry from the `extend::COLORPALETTE` list.

Input None

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextColor
cv_disp iv2
cv_resetColor
cv_nextSymbol
cv_disp iv3
```

cv_resetFillColor

cv_resetFillColor

Action Resets the fill color to white.

Input None

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextColor
cv_disp iv2
cv_resetFillColor
cv_nextSymbol
cv_disp iv3
```

cv_resetLine

cv_resetLine

Action Resets the line style to the default entry that equals the first entry from the `extend::LINEPALETTE` list.

Input None

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextLine
cv_disp iv2
cv_resetLine
cv_nextSymbol
cv_disp iv3
```

cv_resetStyle

cv_resetStyle

Action Resets all curve style attributes such as color, fill color, symbol, and line style to their default values.

Input None

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextSymbol
cv_nextColor
cv_disp iv2
cv_resetStyle
cv_disp iv3
```

cv_resetSymbol

`cv_resetSymbol`

Action Resets the symbol to the default entry that equals the first entry from the `extend::SYMBOLPALETTE` list.

Input None

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextSymbol
cv_disp iv2
cv_resetSymbol
cv_nextColor
cv_disp iv3
```

cv_round

`cv_round cname xdigits ydigits`

Action Rounds off the x-data and y-data values to the specified number of digits.

Input *cname*, name of curve
 xdigits, number of digits to be kept for x-values; default is -1 (no rounding)
 ydigits, number of digits to be kept for y-values; default is -1 (no rounding)

Returns None

Example

```
cv_createFromScript c {1.01 5.05} {9.09 7.07}
cv_round c 1 1
puts "[cv_getVals c]"
=> {1 5.1} {9.1 7.1}
```

cv_scale

`cv_scale cname xm ym`

Action Scales a curve linearly, and replaces x with $xm \cdot x$ and y with $ym \cdot y$.

Input *cname*, name of curve
xm, scale applied to x-values
ym, scale applied to y-values

Returns None

Example

```
# Scales a current over time given in A over s to mA over us.  
cv_scale i_t 1e6 1e3
```

cv_setFillColor

`cv_setFillColor {mode 1}`

Action Switches the fill color on and off.

Input *mode*, sets the fill color:

- 1 fills the symbol with the curve color; default is 1
- 0 specifies the fill color is white

Returns None

Example

```
cv_autoIncrStyle off  
cv_setSymbol 1  
cv_disp iv1  
cv_nextColor  
cv_setFillColor 1  
cv_disp iv2
```

cv_setSymbol

`cv_setSymbol {mode 1}`

Action Switches symbols on and off.

Input *mode*, sets the symbol:

- 1 specifies that the symbols are shown; default is 1
- 0 specifies that the symbols are not shown

Returns None

Example

```
cv_autoIncrStyle off
cv_disp iv1
cv_nextColor
cv_setSymbol 1
cv_disp iv2
```

cv_sort

`cv_sort cname {xdigits 20}`

Action Sorts data points of a curve according to x-values, and removes duplicates.

Input *cname*, name of curve
xdigits, number of digits of x-value to determine whether two values are identical

Returns None

Example

```
cv_createFromScript c {1 2 0.0001 3 0} {2 3 -1 4 1}
cv_sort c 3
puts "[cv_getVals c]"
==> {0 1 2 3} {1 2 3 4}
```

cv_write

`cv_write type filename curveList`

Action Exports curve data to a file. This command works like the native `cv_write` command but, in addition, allows exporting data in CSV format, which is most suitable for transferring data to spreadsheet applications.

Input *type*, type of file: `csv`, `plt`, `xgraph`, or `xmgr`
filename, name of file in which to export data
curveList, list of curves

Returns 1 if successful, 0 otherwise

Example

```
cv_write csv export.csv "idvg cv($n)"
```

dbputs

`dbputs str {dbglevel 1}`

Action Debugs output, where *str* is displayed in the log if the debug variable `::DEBUG` is greater than or equal to the debug level.

Input *str*, string to be printed to standard output
dbglevel, sets the debug level

Returns None

Example

```
dbputs "test1"
set ::DEBUG 2
dbputs "test2"
dbputs "test3" 2
dbputs "test4" 3
set ::DEBUG 0
dbputs "test5"
==> test2, test3
```

ds_getValue

```
ds_getValue proj datasetName {index end}
```

Action Returns the *index*-th value of a dataset.

Input *proj*, project name of the loaded .plt file
datasetName, name of dataset
index, the index of the dataset item to return; counting starts with 0 and finishes with end; default is end

Returns Real number

Example

```
proj_load n5_des.plt
puts "first value [ds_getValue n5_des "anode OuterVoltage" 0]"
puts "last value [ds_getValue n5_des "anode OuterVoltage"]"
```

fi_readTxtFile

```
fi_readTxtFile fname cname {columnIdx 1}
```

Action Reads in data from an ASCII file, where columns are separated by space. The x-values are taken from the first column; the column to be used for the y-values can be specified. The file can contain comment lines starting with a hash character (#).

Input *fname*, name of ASCII file
cname, name of curve to be created
columnIdx, column index to be used for y-values; column-counting starts with 0; default is the second column (*columnIdx*=1)

Returns None

Example

```
fi_readTxtFile "am15g.txt" spec
```

fi_readTxtFileHeader

`fi_readTxtFileHeader fname`

Action Reads and returns the header line – a single line that is neither data nor comment.

Input *fname*, name of ASCII file

Returns List of strings

Example

```
puts "[fi_readTxtFileHeader "aml5g.txt"]"  
==> Wavelength [um] Intensity [W*cm^-2]
```

gr_axis

`gr_axis axis title {xmin ""} {xmax ""} {scale lin}`

Action Sets the attributes of the x-axis and y-axis.

Input *axis*, specifies axis to be modified: x, y, or y2
title, axis label
xmin, *xmax*, sets the range of the axis, where {} specifies automatic scaling
scale, specifies the scaling to apply: lin (default) or log

Returns None

Example

```
gr_axis x {voltage [V]}  
gr_axis y {current [A]} {} {} log
```

gr_resetAxis

`gr_resetAxis`

Action	Resets all the axis attributes and, in particular, switches off the y2-axis.
Input	None
Returns	None

Example

```
gr_axis x {voltage [V]}  
gr_axis y {current [A]} {} {} log  
gr_resetAxis
```

gr_setStyle

`gr_setStyle mode`

Action	Sets the style of the plot area.
Input	<i>mode</i> , specifies the mode: <ul style="list-style-type: none">• "screen" (default) is used for the interactive mode.• "presentation" uses larger font sizes suitable for copying plots into presentations.
Returns	None

Example

```
gr_setStyle "presentation"
```

ldiff

```
ldiff list1 list2 {symmetric ""}
```

Action Returns all items of *list1* that are not in *list2*.

Input *list1*, *list2*, lists to be compared
symmetric, if specified, indicates that the returned list will also contain all items of *list2* that are not in *list1*

Returns List

Example

```
set l1 {1 2 3 4}
set l2 {1 2 5}
puts "[ldiff $l1 $l2]"
==> {3 4}
puts "[ldiff $l1 $l2 1]"
==> {3 4 5}
```

lintersect

```
lintersect list1 list2
```

Action Returns all items that are members of both *list1* and *list2*.

Input *list1*, *list2*, lists to be compared

Returns List

Example

```
set l1 {1 2 3 4}
set l2 {1 2}
puts "[lintersect $l1 $l2]"
==> {1 2}
```

ltranspose

`ltranspose list`

Action Transposes a list.

Input *list*, list to transpose

Returns List

Example

```
set l {{1 2} {3 4} {5 6}}
puts "[ltranspose $l]"
==> {{1 3 5} {2 4 6}}
```

lunion

`lunion list1 list2`

Action Returns a list of unique items that are members of *list1* or *list2*.

Input *list1*, *list2*, lists to be compared

Returns List

Example

```
set l1 {1 2 3 4}
set l2 {1 2 5}
puts "[lunion $l1 $l2]"
==> {1 2 3 4 5}
```

proj_check

```
proj_check proj
```

Action Checks all datasets in a project to see whether all entries are valid.

Input *proj*, project name of the loaded .plt file

Returns List of dataset names containing invalid data (nonnumeric values)

Example

```
proj_load n5_des.plt
proj_check n5_des
```

proj_datasetExists

```
proj_datasetExists proj datasetName {groupName ""}
```

Action Checks whether a project contains data with the specified dataset and group name.

Input *proj*, project name of the loaded .plt file
datasetName, name of dataset
groupName, group name of dataset; if no group name is given and the dataset name contains a space, the first word of *datasetName*, is taken as the group name

Returns 1 (dataset exists) or 0 (dataset does not exist)

Example

```
proj_load n5_des.plt
if {[proj_datasetExists n5_des "anode OuterVoltage"]}
    {puts "anode OuterVoltage exists"}
if {[proj_datasetExists n5_des "OuterVoltage" "anode"]}
    {puts "anode OuterVoltage exists"}
if {[proj_datasetExists n5_des "NO_NODE time"]} {puts "time exists"}
```

proj_getGroups

`proj_getGroups proj`

Action Returns a sorted list containing all group names of the project.

Input *proj*, project name of the loaded .plt file

Returns List of strings

Example

```
proj_load n5_des.plt
puts "all group names: [proj_getGroups n5_des]"
```

proj_groupExists

`proj_groupExists proj groupName`

Action Checks whether a project contains a particular group.

Input *proj*, project name of the loaded .plt file
groupName, name of group

Returns 1 (group exists) or 0 (group does not exist)

Example

```
proj_load n5_des.plt
if {[proj_groupExists n5_des "anode"]} {puts "group anode exists"}
```

proj_loadPlx

```
proj_loadPlx fileName {curveName} {appendDatasetName}
```

Action Opens a .plx file, and creates a curve without displaying it.

Input *fileName*, name of .plx file from which to load data
 curveName, name of curve; if not specified, the dataset name is used
 appendDatasetName, if set to 1, it appends the dataset name to the curve name:

- If *curveName* is empty, then the dataset name is used.
- If a simple curve name is given, then the dataset name is appended in parentheses, for example, `cname(data)`.
- If the curve name contains parentheses, then the dataset name is appended in the parentheses, for example, `cname(cval,data)`.

Returns None

Example

```
Content of test.plx:
"data"
0 0
1 2.8
...

proj_loadPlx test.plx
puts "visible: [cv_isVisible data]"
==> visible: 0
proj_loadPlx test.plx c(1) 1
puts "visible: [cv_isVisible c(1,data)]"
==> visible: 0
```

The PhysicalConstants Library

This library defines a set of variables of major physical constants [\[1\]](#).

To load the library, use the command:

```
load_library PhysicalConstants
```

Table 6 *Variables defined in PhysicalConstants library*

Name of variable	Value	Unit
AtomicMassConstant	1.660540210e-27	kg
AvogadroConstant	6.022136736e23	mol ⁻¹
BohrMagneton	9.274015431e-24	J/T
BoltzmannConstant	1.38065812e-23	J/K
ElectronMass	9.109389754e-31	kg
ElectronVolt	1.6021773349e-19	J
ElementaryCharge	1.6021773349e-19	C
FaradayConstant	9.648530929e4	C/mol
FineStructureConstant	7.2973530833e-3	1
FreeSpaceImpedance	376.730313462	Ω
GravitationConstant	6.6725985e-11	m ³ /kg/s ²
MagneticFluxQuantum	2.0678346161e-15	Wb
MolarVolume	22.4141019e-3	m ³ /mol
Permeability	12.566370614e-7	H/m
Permittivity	8.854187817e-12	F/m
Pi	3.141592653589793	1
PlanckConstant	6.626075540e-34	Js
ProtonMass	1.672623110e-27	kg

Table 6 Variables defined in PhysicalConstants library (Continued)

Name of variable	Value	Unit
RydbergConstant	1.097373153413e7	mol ⁻¹
SpeedOfLight	299792458	m/s
StefanBoltzmannConstant	5.6705119e-8	W/m ² /K ⁴

All variables are defined in the namespace `::const::`. To access a variable, use `::const::<varName>` or `$const::<varName>`, where `<varName>` must be replaced by a particular variable name. For example:

```
load_library PhysicalConstants
puts "c=$const::SpeedOfLight"
```

The function `getVarNames` returns a list of all variable names. For example:

```
set varlist [const::getVarNames]
puts "all variables: $varlist"
==> all variables: AtomicMassConstant AvogadroConstant BohrMagneton ...
```

To see a list of all variables, the function `printVarNames` prints directly the names of all available variables:

```
const::printVarNames
==>
AtomicMassConstant
AvogadroConstant
BohrMagneton
...
```

IC-CAP Model Parameter Extraction Library

The commands of this library are used to export device simulation results to the Integrated Circuit Characterization and Analysis Program (IC-CAP) model extraction tool. These commands can create files that can be later imported by IC-CAP.

To load the library, use the command:

```
load_library ise2iccap
```

Exporting Data

```
iccap_Write fileName headerInfo data
```

Action Exports data to a file using the IC-CAP data management file data format [2].

Input *fileName*, file name
headerInfo, header information (see [Header Information on page 119](#))
data, array of curve data (see [Array Data on page 121](#))

Returns None

Header Information

The header information *headerInfo* is a list formed by the sublists *userInput*, *iccapInput*, and *output*.

A detailed description of the header section is presented in the literature [2]. You can use the following examples as guides.

userInput

This sublist contains information about variables that cannot be swept in a traditional IC-CAP setup.

In the following example, no user sweeps are considered:

```
userInput: {}
```

iccapInput

This sublist contains information about variables that can be swept in an IC-CAP setup. For example:

```
iccapInput: {{vg V G GROUND {LIST 1 26 0.0...2.5}}  
             {vb V D GROUND {LIN 2 -0.1 -0.5 3}}  
             {vd V S GROUND {CON 0.0}}}
```

In this example, there are three IC-CAP input variables, where:

- The first element is the name of the input variable.
- The second element is the mode.

Chapter 8: Working With Script Libraries

IC-CAP Model Parameter Extraction Library

- The third and fourth elements are the names of the positive and negative nodes for the corresponding input variable.
- The fifth element is a list that describes the sweep. The first element of this list is the sweep type, which can be `LIN`, `LIST`, `CON`, `LOG`, or `SYNC`.

The sweep information for the first input variable (`vg`) is a list where:

- `LIST` indicates that all sweep values are explicitly defined.
- `1` is the sweep order (1 is the innermost or fastest varying sweep).
- `26` is the number of values.
- `0.0 . . . 2.5` indicate all values that the particular variable can take.

The sweep information for the second input variable (`vb`) is a list where:

- `LIN` indicates that the sweep values are a set of values defined in a linear scale.
- `2` is the sweep order.
- `-0.1` is the start value.
- `-0.5` is the end value.
- `3` is the number of values.

The sweep information for the third input variable (`vd`) is a list where:

- `CON` indicates that there is only one value for this variable.
- `0.0` is a constant value.

output

This sublist contains information about output variables that can be recognized in an IC-CAP setup. For example:

```
output is {{id I D GROUND}}
```

In this example, there is only one output variable, which is the drain current. The first element is the name of the output variable, the second element is the mode, and the third and fourth elements are the names of the positive and negative nodes for the corresponding output variable.

Array Data

The array data must contain, at least, the following information:

```
data(<input tuples>,<output>)
```

There is one array cell for each pair formed by a tuple of input variable values and an output variable. The `<input tuples>` order is the inverse of the sweep order.

For example, using the example in [Header Information on page 119](#), the array data contains the following information:

```
data(<vb>,<vg>,id)
```

In this case, each cell stores the drain current (`id`) for a particular combination of substrate voltage value (`<vb>`) and gate voltage value (`<vg>`).

For example, the tuple `data(-0.1,1.0,id)` stores the drain current for `vb = -0.1 V` and `vg = 1.0 V`.

Curve Comparison Library

The commands of this library compare two curves by computing the square difference between the two curves within a given domain.

To load the library, use the command:

```
load_library curvecomp
```

cvcmp_CompareTwoCurves

```
cvcmp_CompareTwoCurves curve1 curve2 windowX use_log n
```

Action	Computes the square difference between two curves within a given domain (window) using either linear scale or logarithmic scale.
---------------	--

Input	<code>curve1</code> , <code>curve2</code> , curves to compare <code>windowX</code> , window in the x-axis <code>use_log</code> , true if logarithmic scale is used <code>n</code> , base name for the internal curves
--------------	--

Returns	Square difference between two curves
----------------	--------------------------------------

cvcmp_DeltaTwoCurves

`cvcmp_DeltaTwoCurves exp_file sim_file minX maxX use_log name`

Action Writes the square difference between two curves within a given domain (window) to the standard output. This difference can be computed using either a linear or logarithmic scale. Both curves are read from files. This command uses the `ft_scalar` command to export the computed difference to the Family Tree of Sentaurus Workbench.

Input `exp_file`, `sim_file`, files where the two curves are stored
 `minX`, `maxX`, window in the x-axis
 `use_log`, true if a logarithmic scale is used
 `name`, name of the column of the Family Tree of Sentaurus Workbench where the computed difference is stored

Returns None

References

- [1] G. Woan, *The Cambridge Handbook of Physics Formulas*, Cambridge: Cambridge University Press, 2000.
- [2] *IC-CAP Data Management File Format Specification: Final IC-CAP 5.0 file specification*, E. Arnold and M. Peroolmal (eds.), HP-EESOP document archive, March, 1997.

A

















Elements of User Interface

This appendix lists the toolbar buttons and menus available from the Inspect user interface.

Toolbar Buttons

The toolbar provides quick access to commonly used operations that are also available from the menus.






Table 7 Inspect toolbar buttons

Button	Description	Button	Description
	Loads a dataset file		Shows or hides the grid
	Prints the current plot		Moves selected curve to the front of all curves
	Reloads the dataset file		Moves selected curve to the back of all curves
	Applies recent plotting actions made on the previous dataset to the current dataset		Moves selected curve forward
	Removes all curves, and cleans up the plot area		Moves selected curve backward
	Zooms in to a selected area		Runs or continues executing script
	Zooms out of an area		Stops executing script
	Displays the entire plot area		Switches on or switches off logarithmic scale on the x-axis

Appendix A: Elements of User Interface



File Menu

Table 7 Inspect toolbar buttons (Continued)

Button	Description	Button	Description
	Centers the view in the plot area (applies to a zoomed plot only)		Switches on or switches off logarithmic scale on the left y-axis
	Zooms in to one selected curve		Switches on or switches off logarithmic scale on the right y-axis
	Shows or hides the legend text		

File Menu


Table 8 File menu commands

Command	Toolbar button	Shortcut keys	Description
Load Dataset		Ctrl+L	Opens dataset file.
Update Datasets		Ctrl+U	Reloads datasets from opened files and updates related curves.
Automatically Update Datasets			Automatically reloads datasets from opened files.
Delete Datasets			Deletes selected projects and the curves that use data from them.
Load Setup			Loads preferences stored in setup file.
Save Setup			Saves preferences to setup file.
Restore All			Loads a previously saved project from a .sav file.
Save All			Saves current state of Inspect to a .sav file.
Export			Saves current curves to different file formats.
Write Bitmap		Ctrl+W	Creates a bitmap file of plot area image in PNG format.
Write EPS			Creates an EPS file of plot area image.

Appendix A: Elements of User Interface



Edit Menu

Table 8 File menu commands (Continued)

Command	Toolbar button	Shortcut keys	Description
Write PS			Creates a PostScript file of plot area image.
Print		Ctrl+P	Opens the Printer Setup dialog box.
Preferences			Opens the Preferences dialog box.
Exit		Ctrl+Q	Exits Inspect.

Edit Menu

Table 9 Edit menu commands

Command	Toolbar button	Shortcut keys	Description
Redo Last Plot		Ctrl+E	Applies the last plotting actions to selected datasets.
Plot Area		Ctrl+G	Opens the Plot Area dialog box to change attributes of plot area.
Clean Plot Area			Cleans up the plot area.
Axes		Ctrl+A	Opens the Axes dialog box to change attributes of axes.
Labels			Displays options to add, edit, and remove labels from the plot area.
Define Macros			Opens the Macro Editor.

Curve Menu





Table 10 Curve menu commands

Command	Toolbar button	Shortcut keys	Description
Transform			Displays the following options: Abs X: Maps x-value of all data points of selected curves to its absolute value and redisplay the curve. Abs Y: Maps y-value of all data points of selected curves to its absolute value and redisplay the curve. Reflect X: Reflects curve about x-axis. Reflect Y: Reflects curve about y-axis. Suppress Backtrace: Data points of a selected curve where the x values are not monotonically increasing (where the current x-value is less than the previous one) indicate the start of a new line. In this case, no line connects the previous point to the current point.
Curve Data		Ctrl+D	Opens a dialog box that shows the points of the dataset corresponding to the selected curve.
Restore Data			Undoes all changes to selected curves.
DeltaX (X)			Creates a deltaX curve for each selected curve. The deltaX curve is obtained by taking the x-dataset of the original curve as the x-dataset of the new curve and computing the y-dataset at every point by subtracting the x-value at the current point from the x-value at the next point.
Intersect X ?			Opens a dialog box displaying the x-coordinate at which the selected curve crosses the x-axis. If more than one curve is selected, no action is taken.
Inspector			Opens the Inspector dialog box.

Appendix A: Elements of User Interface




Script Menu

Table 10 Curve menu commands (Continued)

Command	Toolbar button	Shortcut keys	Description
Drawing Order	   		Opens a submenu to rearrange order of curves. Options are: Move to Front: Moves the selected curve to the front of all curves. Move to Back: Moves the selected curve to the back of all curves. Move Forward: Moves the selected curve one step closer to the front. Move Backward: Moves the selected curve one step closer to the back.

Script Menu

Table 11 Script menu commands

Command	Toolbar button	Shortcut keys	Description
Run Script		Ctrl+R	Opens a dialog box to select the script file to be run. The default filter for the script file is *.cmd.
Record			Creates a script file. Options are: Start: Opens the Record Script File dialog box for selecting the output file and starts to record a sequence of operations. Add Pause: Adds a sleep command to the script. The length of the pause is selected from the submenu. Add Break: Adds a break command to the script. Stop: Stops the recording.
Continue Script		Ctrl+C	When a break command is encountered in a script, the execution is suspended and user input is possible. This option reactivates the execution of the script.
Abort Script		Ctrl+N	When script execution is suspended by a break command, this command omits the remaining part of the script.

Extensions Menu

Table 12 *Extensions menu command*

Command	Toolbar button	Shortcut keys	Description
Two-Port Networks		Ctrl+T	Opens the RF Parameter Extraction dialog box.

Help Menu

Table 13 *Help menu command*

Command	Toolbar button	Shortcut keys	Description
About		Ctrl+B	Provides version information.

B

Known Limitations

This appendix describes known limitations that affect working with the Inspect tool.

The `diff(...formula...)` and `integr(...formula...)` Operators

The `diff()` and `integr()` operators require a *curve only* argument, which can be defined as a curve that contains more than one point, since a curve that contains only one point is treated as a scalar. When a formula is used as an argument for these operators, the parser cannot always decide if the argument curve for the `diff()` or `integr()` operators will have more than one point. Therefore, an error message is generated.

For example, to obtain proper results, `diff(log10(...formula...))` must be performed in two steps:

1. Create the curve `log10(...formula...)`.
2. Apply the `diff()` operator to the resulting curve.

The `vecvalx(...formula...)` and `vecvaly(...formula...)` Operators

It is advisable to compute the curve defined by `...formula...` before applying the `vecvalx()` or `vecvaly()` operators. In addition, if the curve displays exponential behavior, better results are obtained if the curve is transformed to logarithmic scale before applying these operators.

For example, suppose you want to compute the value:

```
vecvalx(diff(<c1>), 1e-7)
```

and the curve defined by `diff(<c1>)` has an exponential behavior. In this case, to obtain more precise results, create a curve `<c2>`, which will be equal to `log(diff(<c1>))`, and then compute the required value:

```
vecvalx(<c2>, log(1e-7))
```

No Support for Right Y-Axes

The Inspector dialog box works only for the x-axis and left y-axis.